Trabajo de Final de Máster

Màster en Enginyeria de l'Organització

Cotas y resolución exacta y no exacta del problema de equilibrado de líneas de montaje de tipo I con tareas con deterioro lineal

MEMORIA

Autor: Lorena Angulo Gracia
Director: Alberto García Villoria

Convocatoria: Abril 2019

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resumen

El presente proyecto tiene el propósito de resolver un problema de equilibrado de líneas de montaje de tipo I. Esto significa que el objetivo es minimizar el número de estaciones de trabajo necesarias respetando un tiempo de ciclo dado y cumpliendo con las precedencias y las diferentes restricciones definidas.

A pesar de que este problema está presente en algunas líneas de montaje real, en la literatura no hay numerosos estudios al respecto. Además, la mayoría de ellos consideran que los tiempos de procesamiento de las tareas son constantes, pero esto no siempre es así, ya que en algunos sistemas, el tiempo de una tarea aumenta en función del tiempo que ha transcurrido desde que ha terminado una o varias tareas previas. Entonces, teniendo en cuenta esto y que prácticamente no se ha analizado este concepto, se ha resuelto el problema teniendo en cuenta el efecto del deterioro lineal.

Para ello, se han propuesto dos procedimientos distintos. Por un lado, un modelo matemático que, mediante su programación en el software ILOG CPLEX, se utiliza para resolver los problemas con un número de tareas reducido con el fin de garantizar encontrar una solución óptima en un tiempo finito. Por otro lado, una heurística (método no exacto) basada en la optimización local para abordar los ejemplares más grandes, es decir, los que se asemejan más a la realidad y que por tanto, requieran encontrar la mejor solución en un tiempo breve y de forma sencilla.

También, para poder analizar mejor el comportamiento del deterioro lineal, se han definido cuatro escenarios con una tasa de deterioro distinta en cada uno de ellos: E1 con b=0, E2 con b=0,1, E3 con b=0,2 y E4 con b=aleatorio para cada tarea (donde b significa tasa de deterioro).

Así pues, con el objetivo de evaluar de forma crítica el funcionamiento de los dos procesos de resolución, recientemente descritos, se ha realizado una experiencia computacional por medio del testeo de un conjunto de ejemplares, a los cuales se les ha añadido la tasa de deterioro requerida en cada escenario. Estos están basados en unos ejemplares descargados de una página web que provee datos para distintos problemas de equilibrado de líneas.

Finalmente, se ha han analizado los resultados obtenidos y se ha podido ver que a medida que aumenta la tasa de deterioro en las tareas, como era de esperar, se hace más difícil asignarlas y, en consecuencia, se requiere un mayor número de estaciones necesarias.

Sumario

1. Glosario

ALBP: *Assembly Line Balancing Problem*

***Av$_i$*:** tiempo en el que se encuentra disponible la tarea *i*

***b$_i$*:** tasa de deterioro de la tarea *i*

COSMOAL: *Computer Method of Sequencing Operations for Assembly Lines*

GA: Algoritmos Genéticos

GALBP: *General Assembly Line Balancing Problem*

LINGO: *LINear Generalize Optimizer*

MALBP: *Mixed-Model Assembly Line Balancing Problem*

MOALBP: *Multi-Objective Assembly Line Balancing Problem*

***NP-hard*:** problemas difíciles de resolver

***OS*:** *Order strength* - fuerza de orden

***p$_i$*:** tiempo de proceso total de la tarea *i*

PLEM: Programación Lineal Entera Mixta

PSO: *Particle Swarm Optimization*

SALBP: *Simple Assembly Line Balancing Problem*

SALBP-1 (de tipo I): se quiere minimizar el número de estaciones de trabajo

SALBP-2 (de tipo II): se quiere minimizar el tiempo de ciclo de la línea

SALBP-E: se busca minimizar simultáneamente el número de estaciones y tiempo de ciclo

SALBP-F: una vez dados el tiempo de ciclo y número de estaciones, se trata de determinar si el problema es factible

***S$_i$*:** tiempo de inicio de la tarea i

TC: tiempo de ciclo de la estación

T$_i$: tiempo de procesamiento constante de la tarea *i*

TS: *Tabú Search* – Búsqueda Tabú

UALBP: *U-line Assembly Line Balancing Problem*

VNS: *Variable Neighbourhood Search* - Búsqueda de Vecindad Variable

***W$_j$*:** carga de tiempo de la estación *j*

2. Introducción

Las líneas de montaje son un elemento fundamental en muchos sistemas productivos. Estas consisten en un determinado número de estaciones de trabajo ordenadas a lo largo de, usualmente, una cinta transportadora por la que circula la unidad de producto. En cada una de estas estaciones hay un tiempo de ciclo definido, que determina el tiempo máximo en que los operarios deben realizar sus tareas asignadas antes de que la unidad de producto pase a la estación siguiente.

En el mundo de la industria, existen muchos tipos de líneas de montaje con características diferentes. Por este motivo, a lo largo de los años se han realizado múltiples estudios con el objetivo de entender y conocer su funcionamiento para poder aumentar su productividad y eficiencia. A raíz de esto, se ha podido demostrar que existe un problema de equilibrado en las líneas de montaje, cuyo concepto también se ha analizado profundamente y gracias a ello han ido apareciendo distintos procedimientos para su resolución.

Ahora bien, normalmente, en la literatura se considera que los tiempos de proceso de las tareas son constantes. A pesar de eso, esto no siempre es así, ya que en algunos sistemas, el tiempo de una tarea aumenta en función del tiempo transcurrido desde la finalización de una o varias tareas previas, conociéndose este fenómeno como deterioro. Entonces, ante este contexto, el presente proyecto sí que considera el efecto de deterioro de las tareas.

2.1. Objetivos del proyecto

El objetivo de este proyecto es poder obtener soluciones óptimas resolviendo el problema de equilibrado de líneas de montaje de tipo I, teniendo en cuenta el deterioro lineal de las tareas. Más concretamente, de tipo I significa solventar este problema minimizando el número de estaciones de trabajo necesarias respetando un tiempo de ciclo dado (lo que se conoce como ALBP-1). Es importante destacar que se pretende tener en cuenta el deterioro en la resolución porque es muy importante para poder garantizar la factibilidad del resultado y obtener buenas soluciones.

Por tanto, para llevarlo a cabo, se propone resolver este asunto mediante programación matemática, y para los ejemplares más grandes, de forma heurística. Además, se pretende incorporar cotas para evaluar la calidad de las soluciones encontradas y facilitar la resolución del modelo matemático.

2.2. Alcance del proyecto

El presente trabajo se realiza desde un punto de vista teórico y de investigación, de modo que la resolución del problema se hará en esta línea y no a efectos prácticos.

Para ello, se analizará la literatura existente y se definirá una resolución distinta a las que hay hasta el momento, teniendo en cuenta las siguientes características para el problema tratado:

- La línea de montaje es en serie.
- La tasa de entrada de unidades de producto a la línea de montaje es fija y se procesa un único producto.
- El tiempo de ciclo es un dato que se conoce y es el mismo para cada estación.
- Todas las estaciones son iguales, de modo que cualquier tarea se puede asignar a cualquier estación. Además, en cada estación se permite que su primera tarea no empiece al inicio del ciclo y que haya tiempos ociosos entre tareas.
- Se conoce el número de tareas a realizar en la línea de montaje y el tiempo de procesamiento de cada una de ellas sin deterioro.
- En todo momento se saben las relaciones de precedencia que existen entre las distintas tareas. Cada tarea está disponible siempre y cuando ha finalizado su predecesora/s. Una vez se ha iniciado una tarea, esta no se puede interrumpir, es indivisible.
- El tipo de deterioro que se considera es lineal y se comienza a producir cuando todas las predecesoras han sido realizadas. Es un dato y no tienen por qué tener todas las tareas la misma tasa de deterioro.

Entonces, para comprobar la eficiencia y eficacia de los procedimientos diseñados, se pretende realizar una experiencia computacional con 160 muestras basadas en el conjunto de datos elaborado por *Otto et al.* (2013) para SALBP-1. Estas muestras se van a aplicar en cuatro escenarios distintos, los cuales se diferencian por el valor numérico de la tasa de deterioro y se determinarán más adelante.

3. Estado del arte

3.1. Las líneas de montaje

Las líneas de montaje aparecieron en la era industrial por medio de *Eli Whitney*, quien introdujo el sistema estadounidense de fabricación en serie en 1799, creando piezas estandarizadas que permitían un montaje más rápido y un remplazo mucho más sencillo cuando se rompían. Un siglo más tarde, en 1913, *Henry Ford* hizo realidad este sistema instalando la primera línea de montaje móvil con el objetivo de reducir costes y permitir la producción en masa de automóviles. A partir de este momento, las líneas de montaje se popularizaron en todos los sectores industriales [1]. De hecho, hoy en día, casi todas las empresas de producción utilizan este sistema de fabricación, lógicamente con algunas mejoras posteriores que han permitido hacerlas más flexibles para poder adaptarse constantemente a las necesidades de los consumidores.

Figura 3.1. Imagen de las primeras líneas de montaje de Ford.
[Fuente: https://www.motorpasion.com/industria]

En general, la línea de montaje consiste en un sistema en el que el proceso productivo trata de optimizar los costes, minimizando las pérdidas de tiempo y fomentando la especialización máxima del trabajador y la división del trabajo. También se permite así, crear una maquinaria específica y especializada para cada tarea. Para ello, hay un cierto número de estaciones de trabajo dispuestas a lo largo de una cinta transportadora o de un sistema mecánico similar para que las unidades de producto puedan ir moviéndose de estación a estación por la línea. En cada estación, se realizan repetidamente determinadas operaciones teniendo en cuenta el tiempo de ciclo de la estación, es decir, el tiempo máximo o promedio disponible para cada ciclo de trabajo [2], el tiempo de procesamiento de cada tarea y las relaciones de precedencia o de incompatibilidad entre las distintas tareas.

En definitiva, una línea de montaje tiene como objetivo crear un determinado producto, ya sea final o intermedio. Su implantación ha sido siempre un problema de gran envergadura en la ingeniería industrial y su importancia ha ido creciendo como resultado de la globalización de la competencia, del rápido progreso de las tecnologías de manufactura, del acortamiento de los ciclos de vida de los productos y de la elevada automatización [3]. Es por ello que desde entonces y en la actualidad, las industrias buscan constantemente optimizar sus variables y recursos con el objetivo de reducir costes, mejorar la calidad y eficiencia. Además, el hecho de que la instalación de una cadena de ensamblaje sea una decisión a largo plazo y requiera grandes inversiones de capital, hace necesario que el sistema se diseñe y equilibre para que funcione del modo más eficiente posible [2].

Por este motivo, surge la necesidad de tratar el problema de equilibrado de las líneas de montaje (ALBP), con el fin de asignar las tareas a la secuencia ordenada de las estaciones de tal manera que satisfagan las relaciones de precedencia y se optimice una función objetivo, como por ejemplo, minimizar el tiempo de ciclo, minimizar el número de estaciones necesarias, minimizar el coste global para una tasa de producción determinada, etc. Entonces, según la función objetivo y las restricciones que se definen, existen distintos tipos de ALBP, los cuales se presentan más adelante.

3.1.1. Elementos básicos de los problemas de líneas de montaje

En primer lugar, para poder entender bien los problemas de equilibrado de líneas de montaje y los términos que se utilizan para hablar de su análisis y resolución, es necesario explicar brevemente en qué consisten sus conceptos principales:

- **Tiempo de ciclo (TC):** es el tiempo disponible en cada estación de la línea para completar aquellas tareas que le hayan sido asignadas. En caso de que no se especifique su valor, el tiempo de ciclo de la línea se obtiene considerando el tiempo de la estación que tiene un tiempo de trabajo mayor para realizar sus tareas asignadas.

- **Tarea (*i*):** consiste en una unidad de trabajo indivisible que tiene un tiempo de proceso asociado. La fabricación de una unidad de producto se divide en un conjunto de tareas.

- **Estación:** es una parte elemental y especializada de la línea de montaje. Cada estación está formada por un conjunto de tareas a ejecutar. De modo que la línea de montaje queda constituida por un número de estaciones, dispuestas en serie y/o paralelo, a través de las cuales circula la unidad de producto. Estas pueden estar compuestas por un operador, que puede ser humano o robotizado, por cualquier tipo de maquinaria o por mecanismos especializados.

- **Relaciones de precedencia:** por lo general, se representan con diagramas de precedencias. Son las restricciones sobre el orden en el cual se llevan a cabo las diferentes tareas en la línea. Una tarea cualquiera no se puede ejecutar hasta que no se hayan efectuado todas las que la precedan de forma inmediata.

- **Tiempo de procesamiento de la tarea (t_i):** es el tiempo necesario para realizar esa tarea, el cual depende de las tecnologías de fabricación, de los recursos empleados y de la actividad de los operarios.

- **Tiempo de trabajo de cada estación:** es la suma de los tiempos de procesamiento de todas las tareas asignadas en la estación en cuestión.

- **Tiempo muerto u ocioso de la estación:** para cada estación, es la diferencia entre el tiempo de ciclo y el tiempo de trabajo de dicha estación.

- **Tiempo de inicio de la tarea (s_i):** momento en el que se empieza a realizar cada tarea.

- **Tiempo disponible de la tarea (av_i):** es el instante en el que se puede realizar cada tarea, es decir, cuando todas sus tareas predecesoras han terminado.

- **Deterioro de una tarea (b_i):** es la diferencia entre el tiempo de inicio y el tiempo disponible de la tarea. Por tanto, la tarea se deteriora a medida que espera para ser procesada.

3.1.2. Tipos de problemas de equilibrado de líneas de montaje

Tal y como se ha mencionado previamente, los problemas de equilibrado consisten en distribuir las tareas necesarias para ensamblar la unidad de producto a través de las estaciones que componen la línea de montaje. Según *Baybars*, una línea se considera equilibrada sí, utilizando los recursos al máximo, la suma de los tiempos libres de las estaciones es lo más pequeño posible. Además, si las tareas pueden ser agrupadas de manera que los tiempos de todas las estaciones sean exactamente iguales, se dice que la línea tiene un equilibrio perfecto, lo que en realidad es muy difícil de conseguir [4].

En la literatura se pueden encontrar varias clasificaciones sobre los problemas de equilibrado de líneas de montaje. Cada tipo de problema puede describir una situación muy distinta y particular, y en consecuencia, existen varias modelizaciones dependiendo de los detalles concretos de cada uno de los planteamientos. Ahora bien, según la clasificación propuesta por *Baybars* en 1986, el ALBP (*Assembly Line Balancing Problem*) se puede agrupar en dos grandes categorías: SALBP y GALBP. Acto seguido, se procede a explicarlas con más detalle.

3.1.2.1. SALBP (*Simple Assembly Line Balancing Problem*):

El SALBP consiste en que dada una línea en serie con tasa de entrada fija de unidades de un único producto, se pretende diseñar un conjunto de estaciones de trabajo, cada una de ellas asignada a un operario o máquina, con idéntico tiempo de ciclo o tasa de producción, a partir de un conjunto de tareas con un tiempo de procesamiento preestablecido. De modo que cada tarea sólo es asignable a una estación y éstas pueden presentar entre sí relaciones de precedencia [5]. Además, las tareas son indivisibles y la duración de cada una de ellas es independiente de la estación a la cual se asigna y del orden en el que se procese.

En definitiva, los principales parámetros del problema son el tiempo de ciclo y el número de estaciones, mientras que el resto de datos se conocen. Entonces, según la función objetivo que se desea optimizar este tipo de problema admite distintas variantes [6]:

- **SALBP-1:** se caracteriza por tener un tiempo de ciclo definido (o tasa de producción) y se desea minimizar el número de estaciones de trabajo necesarias para llevar a cabo el proceso. Este tipo es muy común cuando se desea instalar un nuevo sistema de montaje y se pueden obtener datos de la demanda externa.

- **SALBP-2:** se parte de un número de estaciones de trabajo fijas y se quiere minimizar el tiempo de ciclo de la línea. De esta manera se garantizan tiempos mínimos de inactividad. Esto es útil cuando la línea de montaje ya existe.

- **SALBP-E:** se busca minimizar simultáneamente el número de estaciones y tiempo de ciclo, teniendo en cuenta su relación con el tiempo muerto total o la ineficiencia de la línea, es decir, se pretende maximizar la eficiencia de la línea.

- **SALBP-F:** una vez dados el tiempo de ciclo y número de estaciones, se trata de determinar si el problema es factible, y en caso que sí lo sea, encontrar una solución. Es adecuado cuando se quiere saber si la línea de montaje puede trabajar para una combinación cualquiera de tiempo de ciclo y número de estaciones.

3.1.2.2. GALBP (*General Assembly Line Balancing Problem*):

Esta categoría engloba todos los problemas de equilibrado de líneas de montaje que no son SALBP, como por ejemplo, modelos mixtos, estaciones en paralelo, procesamientos alternativos, tiempos de proceso variables, etc. En definitiva, esta tipología se caracteriza por dar cabida a problemas más reales y cuotidianos. Entre los principales se encuentran los siguientes problemas GALBP [7]:

- **UALBP (*U-line Assembly Line Balancing Problem*):** se caracterizan por, en vez de utilizar una línea en serie, trabajar con una en forma de U. En una línea en forma de U, las estaciones se pueden colocar de manera que se pueda manejar a la vez dos unidades de producto en diferentes posiciones de la línea. Por tanto, en este tipo hay un mayor número de posibilidades de asignar las tareas a las estaciones, por lo que se puede resolver el problema de forma más eficiente que en una línea en serie.

- **ALBP con deterioro:** se considera que el tiempo de procesamiento total para realizar una tarea no es constante, sino que varía en función de la estación a la cual se asigna y del orden en el que se procesa. Esto se debe a que la duración de una tarea aumenta según el tiempo que transcurre desde que terminan sus predecesoras hasta que se inicia dicha tarea. Se define una tasa de deterioro, la cual puede ser la misma o distinta para cada tarea.

- **MALBP *(Mixed-Model Assembly Line Balancing Problem)*:** se trata del equilibrado de líneas con modelos mixtos, el cual se basa en tener varios modelos de un mismo producto, teniendo un conjunto común de tareas a realizar en la totalidad de los modelos y sin tener en cuenta los tiempos de preparación de la línea.

- **RALBP (*Robotic Assembly Line Balancing Problem)*:** consiste en el equilibrado de líneas robotizadas. Aquí se desea optimizar la ejecución de las tareas en la línea, considerando tanto la asignación de las tareas a cada una de las estaciones como la destinación de cada uno de los robots a las diferentes estaciones.

- **MOALBP *(Multi-Objective Assembly Line Balancing Problem)*:** es el equilibrado de líneas con objetivos múltiples. Se caracteriza por buscar varios objetivos a la vez, combinando, por ejemplo, minimizar el número de estaciones y maximizar la eficiencia.

- **Otros:** como el FTALB (*Flexible task Time Assembly Line Balancing*), TSALBP (*Time and Space constrained Assembly Line Balancing Problems*) y PALBP (*Parallel Assembly Line Balancing Problem*), aunque la información existente acerca de ellos es escasa.

Es importante destacar que en cada uno de los GALBP presentados, también hay distintas variantes del problema según la función objetivo, igual que en los SALBP, es decir, pueden ser de tipo 1, tipo 2, tipo E o tipo F.

A continuación, se muestra el resumen de los tipos de problema de equilibrado de líneas de montaje para ver de forma rápida y clara lo que se acaba de explicar:

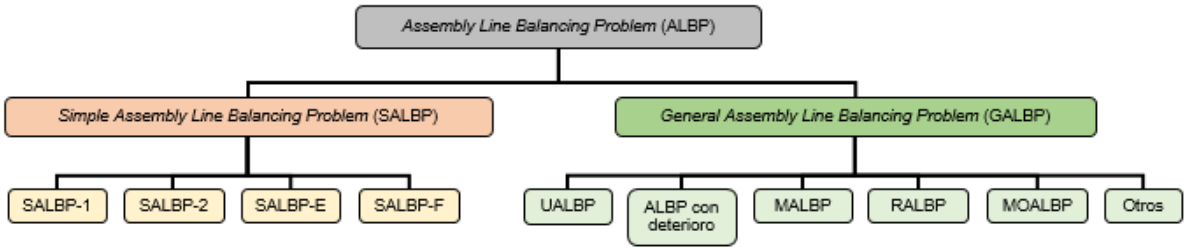


Figura 3.2. Clasificación de los tipos de problemas de equilibrado de líneas (ALBP). [Fuente: propia]

3.1.3. Procedimientos de resolución de problemas de equilibrado de líneas

A lo largo de los años, un gran número de autores han expuesto diferentes modelos de resolución para los problemas de equilibrado de líneas, tanto del tipo SALBP como GALBP.

Por un lado, en cuanto a los problemas de equilibrado de líneas de montaje generales (GALBP), hay que destacar que los trabajos existentes son muy variados y con metodologías distintas según las características del problema concreto. Una de las mejores obras que expone los algoritmos de resolución de GALBP es la de *Becker* y *Scholl* (2006). Por otro lado, sobre los SALBP también se ha podido encontrar una gran cantidad de algoritmos desarrollados.

Entonces, en resumen, en ambos casos los procedimientos de resolución se pueden agrupar en dos métodos, los métodos exactos y los métodos heurísticos, aunque no se ha estudiado cuál de los dos es la mejor forma de modelizar y cuál es la mejor resolución.

3.1.3.1. Métodos exactos

Los métodos exactos garantizan encontrar una solución óptima en un tiempo finito, siempre y cuando esta exista, usando programación matemática y algoritmos exactos para la exploración de grafos (representación gráfica de diversos puntos, nodos o vértices).

Para entrar más en detalle, a continuación, se exponen los procedimientos exactos que habitualmente se utilizan.

- **Programación matemática:**

La programación lineal (PL) da respuesta a situaciones en las que se desea maximizar o minimizar una función objetivo lineal en varias variables reales, que están sujetas a determinadas restricciones (expresadas por inecuaciones lineales). En definitiva, todas las expresiones son lineales y se distinguen 3 tipos de programación lineal: cuando todas las variables son enteras (PLE), cuando todas son binarias (PLB), y cuando hay algunas binarias y otras enteras (PLEM).

Un ejemplo de este tipo de procedimiento aplicado a resolver el problema ALBP, es el algoritmo de *White* (1961), o bien, el ideado por *Valero* (1991).

- **Exploración dirigida o *Branch and Bound* (B&B):**

Este método organiza las soluciones en paquetes, progresivamente más pequeños, y determina para cada paquete un indicador de la calidad de las soluciones que contiene. El indicador resultante permite considerar qué subespacio de soluciones es el más interesante para explorar, y una vez se determina, se lleva a cabo la exploración, la cual consiste en dividir el subespacio por dos o más subespacios.

Algunos ejemplos de autores que propusieron este tipo de procedimiento para la resolución del problema de líneas de montaje son: *Hoffman* (1992) con el algoritmo EUREKA, y *Klein y Scholl* (1997) con el algoritmo SALOME.

- **Programación dinámica:**

Con este método no se cuenta con una formulación matemática estándar, sino que se trata de un enfoque general para la solución de problemas. De manera que las ecuaciones específicas que se utilizan, se deben desarrollar para que representen cada situación individual. Con esto se consigue resolver el problema por etapas, en cada una de las cuales se puede tomar una decisión independiente de las decisiones consideradas con anterioridad.

Se necesita cierto grado de ingenio y un buen conocimiento de la estructura general de los problemas de programación dinámica para reconocer cuando y como resolver un problema por medio de estos procedimientos [8]. A pesar de ello, hay varios autores que han aplicado la programación dinámica para la resolución del problema de equilibrado de líneas, como por ejemplo, la obra de *Held*, *Karp* y *Shareshian* (1963).

3.1.3.2. Métodos heurísticos

Los métodos heurísticos son algoritmos que suelen dar, en un tiempo breve y de forma simple, una buena solución, aunque sin garantizar siempre la óptima buscada. Por este motivo, estos se suelen utilizar para acelerar el proceso de encontrar una solución satisfactoria. Especialmente se emplea en los problemas de tipo *NP-hard*, es decir, en los problemas en los cuales no se puede garantizar encontrar la mejor solución en un tiempo razonable.

Están construidos sobre el uso de diversos procesos empíricos, es decir, sobre estrategias que se basan en la experiencia, práctica y observación de los hechos, con el objetivo de lograr encontrar la solución eficaz del problema determinado.

Por un lado, en la literatura existente, se encuentra la obra de *Talbot*, *Patterson* y *Gehrlein* (1986), la cual hace una clasificación de los algoritmos heurísticos que hasta el momento del estudio servían para resolver el ALBP, siendo su categorización la siguiente [9]:

- **Heurísticas de composición:**

Corresponden a una composición de reglas de decisión. Entre estas heurísticas destaca el algoritmo COMSOAL de *Arcus* (1966).

- **Reglas de *backtracing* (retroceso):**

Estas reglas utilizan una técnica de programación, la cual tiene diferentes caminos a elegir. Entonces, una vez se ejecuta todo el proceso, si la solución encontrada no cumple con las condiciones dadas, se vuelve hacia atrás para buscar otro camino que permita que la nueva solución que se encuentre pueda ser óptima, y así, sucesivamente.

En este caso, los principales algoritmos son la heurística de *Hoffman* (1963) y el algoritmo de MALB, desarrollado por de *Dar-El* (1973).

- **Aproximación partiendo de algoritmos exactos:**

Estas heurísticas parten de un procedimiento exacto al cual se le limita el tiempo de búsqueda de la solución óptima. Aquí el objetivo es encontrar soluciones de calidad y que sus tiempos de ejecución estén acotados por cotas conocidas. Los algoritmos destacados son el de *Held*, *Talbot y Patterson* (1986), y el algoritmo de FABLE de *Johnson* (1988).

- **Heurísticas voraces o *greedy*:**

Es un algoritmo constructivo orientado a resolver problemas con más de un recurso. Se caracteriza por utilizar datos disponibles del problema para construir paso a paso una solución del mismo. Para ello, se parte de una lista ordenada de tareas según un índice de prioridad, y luego se secuencian y temporizan las tareas, respetando las restricciones de precedencia y la limitación de recursos. Una vez se han programado todas las tareas, se obtiene una solución de forma rápida, aunque no tiene por qué ser la óptima.

Es una de las heurísticas más conocidas, prueba de ello es que los investigadores utilizan constantemente este tipo de algoritmo para el desarrollo de nuevos procedimientos. Entre otras, se puede encontrar en la obra de *Preecha* y *Nuchsara Kriengkorakot* (2016), y la de Burgos-Meneses, Garzón-Aguirre y López-Pereira (2013).

También existen casos particulares de las *greedy,* como por ejemplo, la heurística de una sola pasada. Los algoritmos más conocidos son el de *Tongue* (1960), el de *Helgeson & Birnie* (1961), y el de *Moodie* y *Young* (1965).

Por otro lado, en la actualidad, también se pueden encontrar otras metaheurísticas que se han ido proponiendo a lo largo de los años como procedimientos de resolución. Su gran versatilidad y simplicidad a la hora de manejarlas, ha hecho que se hayan popularizado cada vez más para resolver este tipo de problemas.

Antes de concretar algunos ejemplos, es necesario saber que las metaheurísticas son procedimientos que permiten tanto la exploración, para la búsqueda en un gran espacio de soluciones factibles, como la explotación, para concentrar la búsqueda en una región reducida del espacio factible donde puede estar la solución óptima. A diferencia de las heurísticas, que se basan en procedimientos simples que utilizan el sentido común

A continuación, se explican algunas de las metaheurísticas más conocidas que se han ido proponiendo hasta el momento para resolver el problema de equilibrado de líneas.

- **Algoritmos genéticos (GA):**

  Son algoritmos de optimización, búsqueda y aprendizaje que se basan en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, tal y como postuló Darwin en 1859. Así pues, imitando este proceso, los algoritmos genéticos son capaces de ir creando soluciones para problemas del mundo real [10].

  Básicamente, consisten en una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuáles de ellos deben generar descendencia para la nueva generación. Se trata de una técnica robusta que puede tratar con éxito una gran variedad de problemas, incluyendo aquellos en los que otros métodos encuentran dificultades. En la literatura, alguna de las obras que lo aplican son las de *Rubinovitz y Levitin* (1995) y *Noushabadi* (2011).

- **Búsqueda tabú (TS):**

  Es un procedimiento adaptativo con la capacidad de hacer uso de muchos otros métodos, como algoritmos de programación lineal y heurística especializada, con el objetivo de superar las limitaciones de la optimización local [11]. Consiste en una búsqueda local que trata de no quedar atrapada en un óptimo local, mediante el uso de información que recauda a medida que el algoritmo transcurre. Por esta razón, es uno de los enfoques más representativos de los algoritmos con memoria, ya que extrae información de lo sucedido y actúa en consecuencia, manteniendo una lista tabú que le impide volver a soluciones ya visitadas.

Se encuentra como ejemplo, la obra de Pastor, Andrés, Duran y Pérez (2002), donde exponen este algoritmo para la resolución de un caso particular de ALBP.

- **Búsqueda Local (*Local Search*):**

Este método se trata de un proceso iterativo que parte de una solución inicial factible y a partir de esta se generan soluciones vecinas, las cuales se pueden obtener, por ejemplo, por medio de alguna heurística o de forma aleatoria. De modo que se basa en que las soluciones buenas se encuentran cercanas entre sí y, en consecuencia, comparten una estructura común.

Entonces, si se encuentra que alguna de las soluciones vecinas es mejor que la solución inicial, automáticamente esta pasará a ser la nueva solución inicial y se iterará para buscar soluciones vecinas sobre ella. Este procedimiento se va repitiendo hasta que no se encuentra ninguna vecina próxima que sea mejor que la solución que se tiene. Un ejemplo que utiliza un procedimiento similar es la obra de *Sener Akpinar* (2017), donde se emplea el algoritmo LNS para resolver el problema de equilibrado de líneas de tipo II.

- **Colonia de hormigas (ACO, *Ant Colony Optimization*):**

Estos algoritmos se caracterizan por simular el comportamiento de las hormigas cuando forman las rutas entre su nido y su fuente de alimento, en base al rastro de las feromonas que van depositando en la trayectoria que realizan. De manera que el principio de este heurístico se basa en que la probabilidad de que una hormiga seleccione una ruta depende del número de hormigas que la hayan seleccionado previamente y de su distancia. Gradualmente, las hormigas preferirán usar las rutas más cortas. El concepto de mínima distancia entre el nido y la fuente de alimento se adaptará a la función de aptitud, cuyo mínimo valor indicará la ruta a seleccionar, por tanto, en este algoritmo se construyen soluciones iterativamente [12].

Uno de los trabajos que aplica este tipo de algoritmo para la resolución del problema de equilibrado de líneas es el de *Baykasoglu*, *Dereli*, *Erol* y *Sabancu* (2003).

- **Optimización por enjambre de partículas (PSO, *Particle Swarm Optimization*):**

Está inspirado en el comportamiento social de una bandada de aves migratorias que intentan llegar a un destino desconocido. De modo que este algoritmo permite optimizar el problema a partir de una población de soluciones candidatas, las cuales va moviendo por todo el espacio de búsqueda según las reglas matemáticas, que tienen en cuenta la mejor posición hallada hasta el momento y la velocidad de cada una de las partículas.

Una partícula es análoga a un cromosoma en el algoritmo genético (GA), con la diferencia de que el proceso evolutivo en el PSO no crea nuevas aves a partir de las originales, sino que las aves en la población solo desarrollan su comportamiento social y, en consecuencia, su movimiento hacia un destino [13].

Algunas de las obras existentes en la literatura que emplean este procedimiento para el SALBP son la de *Qi Lv* (2011), y *Petropoulos* y *Nearchou* (2015).

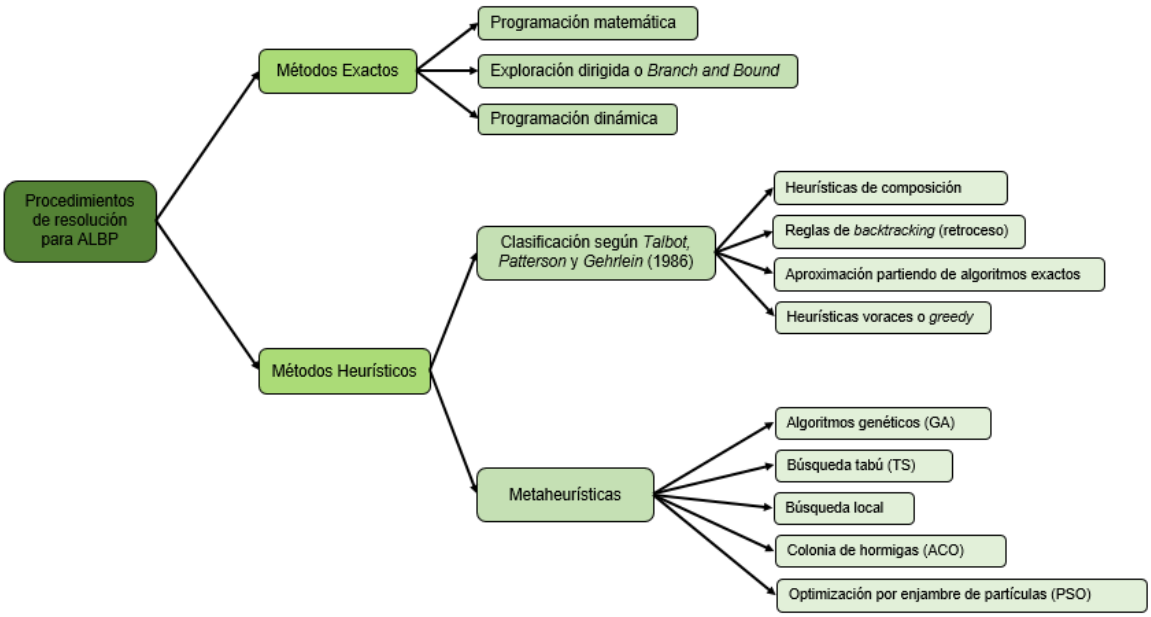


Figura 3.3. Esquema resumen de los algoritmos de resolución para ALBP. [Fuente: propia]

En conclusión, es importante quedarse con la idea de que los métodos exactos garantizan una solución óptima (si es que existe), pero que por lo general, su inconveniente es que sólo pueden utilizarse en problemas con un número reducido de tareas, ya que el problema de equilibrado de líneas es *NP-hard* y la resolución exacta, en general, de ejemplares no pequeños requeriría de un tiempo de cálculo demasiado elevado en la práctica. Por este motivo, para poder abordar estos problemas complejos de dimensiones reales, han aparecido muchos métodos heurísticos de resolución eficiente que aunque no garanticen siempre la solución óptima, suelen dar una solución de calidad en poco tiempo y de forma sencilla.

Ahora bien, una vez explicados los métodos de resolución, también hay que añadir que hay una gran cantidad de herramientas que permiten implementar estos procedimientos. Entre las más utilizadas, destacan las siguientes:

- **LINGO (*LINear Generalize Optimizer*):** herramienta diseñada para construir y resolver modelos de optimización matemática que permite formular problemas lineales y no lineales. Permite tener modelos que son fáciles de mantener [14].

- **ILOG CPLEX:** es una familia de herramientas analíticas de decisión, para el rápido desarrollo e implementación de modelos de optimización, que utiliza programación matemática y programación por restricciones. Combina un entorno de desarrollo integrado (IDE, por su sigla en inglés), con un poderoso Lenguaje de Programación de Optimización (OPL) y *solvers* de optimización ILOG CPLEX de alto desempeño.

- **GAMS (*General Algebraic Modeling System*):** está diseñado específicamente para modelar problemas de optimización tanto lineales, no lineales y mixtos, así como realizar programación matemática. El sistema es especialmente útil para solucionar problemas que sean grandes y complejos que puedan necesitar muchas revisiones antes de establecer el modelo final.

- **COSMOAL (*Computer Method of Sequencing Operations for Assembly Lines*):** se trata de una heurística definida por *Arcus* (1966), creada para dar solución al problema de equilibrado de las líneas de montaje, aunque también puede abordar otros tipos de problemas. Este método crea rápidamente una lista de tareas a programar, de tal manera que solo aparecen aquellas que son factibles. De forma aleatoria se elige la tarea a asignar a la estación hasta generar la secuencia, y las mejores secuencias se convierten en límites superiores. [15].

- **NEOS *Server for Optimization*:** es un servicio gratuito basado en Internet para resolver problemas de optimización numérica. Proporciona acceso a más de 60 solucionadores de vanguardia, en más de una docena de categorías de optimización, en las que se incluyen la programación lineal, programación entera y optimización no lineal [16].

- **Lenguajes de programación:** como *Python* y *Visual Basic* para programar heurísticas y metaheurísticas.

3.2. Efecto del deterioro en las tareas

Por lo general, en la literatura de investigación de las líneas de montaje, los estudios realizados han considerado que el tiempo de procesamiento de las tareas es un valor constante, que no cambia con el tiempo. De manera que, dentro de cada estación, han supuesto que las tareas se pueden realizar en cualquier secuencia factible sin que afecte a su tiempo de proceso, es decir, sin deterioro.

Sin embargo, en algunas situaciones del mundo real, la tarea se procesa más tarde de su inicio, y, por tanto, consume más tiempo del necesario para llevarse a cabo. Este efecto es el que se conoce como deterioro y fue introducido por primera vez por *Gupta* y *Gupta* (1988), y *Browne* y *Yechiali* (1990), los cuales propusieron modelos en los que el tiempo de procesamiento de una tarea se basa en una función lineal de su tiempo de inicio.

Un ejemplo de este fenómeno se puede expresar suponiendo que una tarea consiste en pintar una pieza y otra tarea posterior en agujerear dicha pieza. La duración de la tarea de agujerear dependerá del intervalo transcurrido desde que se pintó hasta que la pintura esté seca, la cual puede variar en función de distintas circunstancias. En consecuencia, la tarea de agujerear la pieza puede verse afectada y empezar a procesarse más tarde de su tiempo de inicio definido.

Entonces, al considerar que este problema puede ocurrir en muchos casos, es necesario determinar un cronograma y una secuencia razonable de las tareas en las estaciones de trabajo teniendo en cuenta el posible deterioro, para evitar una reducción de la productividad que conlleve a costes extras significativos.

Actualmente, hay pocos estudios que consideren el deterioro de la tarea en el problema de equilibrado de líneas de montaje con el enfoque de la programación. El primero en hacerlo fue *Toskari et al*. (2010), y le siguieron otros autores, tal y como se analizará en el capítulo siguiente.

4. ALBP con tareas con deterioro

A continuación, en este apartado se describe el problema en concreto que se va a tratar en el presente trabajo y los procedimientos diseñados para solventarlo. Para ello, en primer lugar, se hace una revisión de la literatura existente, luego se comentan sus principales características y especificaciones, y finalmente se definen y formalizan los procedimientos diseñados para su resolución.

4.1. Estado del arte

Antes de empezar, hay que destacar que, como bien ya se ha comentado, aunque el efecto de deterioro de la tarea es un concepto muy importante en las líneas de ensamblaje, ha habido pocos trabajos de investigación que tengan en cuenta este aspecto. Por este motivo, este proyecto pretende estudiar el problema de equilibrado de líneas de montaje (ALBP) considerando el efecto de deterioro.

Como se ha especificado en el capítulo anterior, el deterioro de la tarea se empieza a producir desde que la tarea está disponible hasta el instante en el que se comienza a procesar. Después de revisar la literatura existente, se han encontrado ocho referencias que también tienen en cuenta el deterioro de las tareas en el ALBP.

Así pues, analizando en orden cronológico, uno de los primeros artículos que se encuentra es el de ***Toskari et al.*** **(2010a)**, el cual se basa en el problema de SALBP de tipo I, es decir, en minimizar el número de estaciones para un tiempo de ciclo dado. Además, también tuvo en cuenta efectos simultáneos de aprendizaje y deterioro lineal para todas las estaciones. Para resolverlo, propuso un modelo de formulación mixta de programación entera no lineal, que ejecutó mediante la herramienta LINGO, y para los ejemplares más grandes del problema propuesto empleó un enfoque COSMOAL.

El mismo año, ***Toskari et al.*** **(2010b)** también publicaron otra obra acerca del problema de SALBP de tipo I, pero esta vez bajo cuatro combinaciones conjuntas: dos efectos de aprendizaje y dos efectos de deterioro. Ambos efectos también se consideran simultáneamente. Más concretamente, los dos efectos de aprendizaje que consideró son el dependiente de la posición de *Biskup* (1999) y el dependiente del tiempo propuesto por *Kuo* y *Yang* (2006). Mientras que los dos efectos de deterioro que supuso son el de que el tiempo de procesamiento de un trabajo es una función lineal de su hora de inicio (*Mosheiov*, 1996), y el deterioro no lineal, propuesto por *Alidaee* y *Womer* (1999). Esta vez, para su resolución empleó una heurística *greedy*.

Antes de seguir, es necesario definir que el concepto de efecto de aprendizaje se basa en el hecho de que una actividad puede realizarse de forma más eficiente a medida que se incrementa el número de repeticiones de la misma.

Por un lado, otro documento a tener en cuenta es el de ***Shahanaghi et al.*** **(2010)**, el cual introduce el deterioro lineal de la tarea en el SALBP analizando el objetivo de tipo II, es decir, buscando una asignación óptima y un cronograma de las tareas en las estaciones de trabajo para minimizar el tiempo de ciclo para un número dado de estaciones. Para los problemas simples propuso un modelo matemático de PNLEM, que resolvió mediante el software LINGO, y un algoritmo genético para ejemplares más grandes.

También hay que considerar un estudio muy similar al de *Shahanaghi et al.* (2010), el de ***Noushabadi et al.*** **(2011)**. Este último se centra en el problema de SALBP de tipo I, a diferencia de *Shahanaghi et al.* (2010) que analiza el de tipo II. Ahora bien, en ambos casos proponen un modelo matemático de PNLEM y un algoritmo genético similar, con una nomenclatura y unos resultados bastante parecidos entre ellos. Además, se ha observado que en los dos casos aparecen los mismos tipos de errores detectados, por ejemplo, presentan valores incorrectos de los tiempos de proceso de las mismas tareas y ecuaciones iguales formuladas de forma errónea.

Por otro lado, está la obra de ***Bahalke et al.*** **(2011)**, la cual se aleja un poco de la visión clásica de los autores anteriores e introduce el caso de minimizar el tiempo de flujo bajo los efectos de deterioro de la tarea para el problema simple de equilibrado de líneas de montaje. El tiempo de flujo es la suma de los tiempos de procesamiento de todas las tareas en el total de estaciones de la línea. Formula un modelo matemático de PNLEM que ejecuta por medio de la herramienta LINGO y resuelve varios ejemplares conocidos aplicando una heurística *greedy* propuesta.

Otro artículo a tener presente es el de ***Karimi-Nasab et al.*** **(2012)**, en el cual se analiza el efecto de deterioro lineal de las tareas en las líneas de ensamblaje, estudiando concretamente el objetivo de minimizar el tiempo de trabajo, es decir, pretende producir soluciones viables teniendo en cuenta el tiempo de ciclo dado y el número de estaciones de trabajo, simultáneamente. De manera que para su resolución desarrolla un modelo matemático y un algoritmo genético, este último para emplearlo cuando se desea obtener resultados rápidos en los ejemplares más grandes que son difíciles de resolver.

Por el contrario, ***Shabani*** **(2013)** analiza el problema de equilibrado de líneas de montaje configuradas en forma de U, a diferencia del resto de los autores anteriores que consideran la forma tradicional, es decir, que en las líneas las estaciones se organizan a lo largo de una línea recta. El problema se basa en el objetivo de tipo II, teniendo también en cuenta el efecto de deterioro de las tareas para evitar que por este motivo aumente el tiempo de ciclo. Para

resolverlo propone una formulación de programación no lineal entera mixta y dos metaheurísticas. Estas dos últimas se tratan de un algoritmo genético y una optimización de enjambre de partículas (PSO) para resolver el problema en un período de tiempo razonable.

Como última referencia encontrada, está la de ***Hamta et al.*** **(2014)**. Este documento aborda el problema de equilibrado de líneas de ensamblaje de tipo II con efectos simultáneos de deterioro y aprendizaje, en los que hay tiempos de configuración independientes de la secuencia relacionada con cada tarea. Se desarrolla un modelo matemático (PNLEM) para este problema, donde se presenta la programación de ejecución de las tareas asignadas a cada estación de trabajo. Además, propone un método metaheurístico híbrido para resolver el problema cuando se vuelve complejo, en el cual utiliza la búsqueda tabú dentro de la búsqueda de vecindad variable (VNS/TS).

A continuación, una vez descritas las referencias, es importante destacar que en los trabajos de *Shahanaghi et al.* (2010), *Bahalke* (2011), *Noushabadi et al.* (2011), y *Karimi-Nasab et al.* (2012), se considera que una tarea está disponible en el instante en el que se acaban de realizar todas sus tareas predecesoras, aunque, no se aclara si las tareas que no tienen ninguna predecesora tienen o no deterioro. Además, se ha comprobado (en los tiempos de procesamiento de las soluciones que muestran) que todos estos artículos no tienen en cuenta los tiempos ociosos (muertos) de las estaciones de trabajo, es decir, los instantes en los que cada estación, dentro del tiempo de ciclo correspondiente, no procesa ninguna tarea. Además, se debe añadir que el autor *Bahalke* también fue uno de los participantes en la obra de *Noushabadi et al.* (2011), y *Karimi-Nasab et al.* (2012).

En cambio, en los artículos de *Toskari et al.* (2010a, 2010b), *Shabani* (2013) y *Hamta et al.* (2014), se ha podido ver que para cada estación, todas las tareas asignadas a dicha estación están disponibles al inicio del ciclo. De manera que una tarea que se realice al final de una estación tendrá mucho deterioro, pero en cambio, si se lleva a cabo en la estación siguiente a esta, ya no tendría deterioro. Esto no tiene mucho sentido, ya que el deterioro debería seguir estando.

Entonces, a modo resumen, para poder ver de forma clara la comparativa entre las referencias que se acaban de comentar, se muestra en la Tabla 4.1 sus características principales:

Tabla 4.1. Comparativa estado del arte encontrado para el ALBP con deterioro en las tareas.

	A	B	C	D	E	F
Toskari et al. (2010a)	Tipo I (SALBP-1)	Sí	2	Lineal	Sí	PNLEM (LINGO) y COSMOAL
Toskari et al. (2010b)	Tipo I (SALBP-1)	Sí	2	Lineal y No Lineal	Sí	Heurística *greedy*
Shahanaghi et al. (2010)	Tipo II (SALBP-2)	No	1	Lineal	No	PNLEM (LINGO) y GA
Bahalke et al. (2011)	Dado el tiempo de ciclo y nº de estaciones, [MIN] el tiempo total de proceso	No	1	Lineal	No	PNLEM (LINGO) y Heurística *greedy*
Noushabadi et al. (2011)	Tipo I (SALBP-1)	No	1	Lineal	No	PNLEM y GA
Karimi-Nasab et al. (2012)	Dado el tiempo de ciclo y nº de estaciones, [MIN] el tiempo total de proceso	No	1	Lineal	No	PNLEM (LINGO) y GA
Shabani (2013)	Tipo II (UALBP-2)	No	2	Lineal	No	PNLEM, GA y PSO
Hamta et al. (2014)	Tipo II	Sí	2	Lineal	Sí	PNLEM (LINGO) Híbrido VNS +TS

A: Objetivo
B: ¿Incluye el efecto de aprendizaje?
C: Las tareas están disponibles al acabar sus predecesoras (1) o, para cada estación, al inicio de ciclo (2)
D: Tipo de deterioro
E: ¿Tienen todas las tareas la misma tasa de deterioro?
F: Procedimiento de resolución

Por último, comentar que tal y como se puede observar en la Tabla 4.1, las cuatro referencias que incluyen el efecto de aprendizaje consideran que todas las tareas tienen la misma tasa de deterioro. Por el contrario, las referencias que no incluyen este efecto, optan por no asignar la misma tasa de deterioro a cada una de las tareas.

4.2. Definición del problema tratado

Tal y como se ha ido comentado, el equilibrado de líneas de montaje es un problema muy importante en los sistemas productivos, ya que la producción depende directamente de cómo se asignen las diferentes tareas a las estaciones de trabajo disponibles. La programación de las tareas en el mundo industrial se ha planteado de muchas maneras diferentes y se ha resuelto con gran variedad de métodos, ya sean exactos o heurísticos.

En este apartado se procede a definir el objetivo y las especificaciones concretas que se han planteado para abordar la resolución de este problema.

De modo que, dada una línea en serie con una tasa de entrada fija de unidades de producto, el problema se basa en asignar y programar *N* tareas conocidas en distintas estaciones de trabajo, teniendo en cuenta sus relaciones de precedencia (las cuales vienen dadas) y tiempo de procesamiento, respetando siempre el tiempo de ciclo determinado, el cual es el mismo para cada estación. Por tanto, se trata de un **ALBP-1**, ya que se pretende minimizar el número de estaciones necesarias en una línea de montaje. Además, se considera el deterioro lineal de las tareas. Este último concepto se puede definir como el tiempo que pasa entre el momento en que una tarea está disponible para realizarse y el tiempo en el que verdaderamente se inicia. Así pues, ante esta situación, el tiempo de procesamiento de cada tarea teniendo en cuenta su posible deterioro se calcula de la forma siguiente:

$$p_i = t_i + b_i \cdot (st_i - av_i) \qquad \text{(Ec. 4.1)}$$

donde para cada tarea *i*, se tiene que $\boldsymbol{t_i}$ es la parte constante del tiempo de proceso dado sin deterioro, $\boldsymbol{b_i}$ es la tasa de deterioro, $\boldsymbol{st_i}$ es el tiempo en el que se inicia la tarea, y $\boldsymbol{av_i}$ queda definido por el instante en el que la tarea ya está disponible para realizarse. Hay que tener en cuenta que cada tarea está disponible para ejecutarse siempre y cuando haya finalizado su/s predecesora/s.

A continuación, para entender bien los conceptos que se acaban de explicar, se muestra la resolución de un ejemplo pequeño, en el cual se debe determinar el número de estaciones necesarias para llevar a cabo 5 tareas en una línea de montaje. Los datos que se conocen son los siguientes:

- El tiempo de ciclo (TC) de cada estación es de 10 segundos
- La tasa de deterioro (b_i) se considera que tiene un valor de 0,1 y es común para todas las tareas
- El tiempo de procesamiento (t_i) de cada una de las tareas es:

Tabla 4.2. Tiempo de proceso de cada tarea del ejemplo presentado.

Tarea *i*	t_i
1	3
2	6
3	7
4	1
5	5

- Las relaciones de precedencia entre tareas son las siguientes:

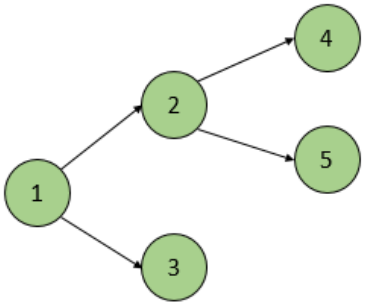


Figura 4.1. Grafo de las relaciones de precedencia del ejemplo presentado. [Fuente: propia]

Una vez presentados los datos del ejemplo, se pueden encontrar diferentes soluciones factibles. Una de ellas podría ser la mostrada en la Tabla 4.3, en la cual el criterio que se ha utilizado para colocar las tareas en las estaciones es el orden numérico ascendente. En la tabla se puede observar que cuando la columna de st_i y av_i no coinciden, quiere decir que la tarea sufre deterioro, ya que no se inicia cuando ha finalizado su/s predecesora/s, si no más tarde. También se puede ver, que en este caso, ninguna estación trabaja al tiempo de ciclo máximo (10 segundos) que podría, lo que significa que el tiempo que trabajan es menor.

Tabla 4.3. Cálculo de una solución obtenida para el ejemplo propuesto de 5 tareas.

Nº estación	Tarea *i*	st_i (s)	av_i (s)	p_i (s)	Tiempo en la estación (s)
1	1	0,00	0,00	3,00	9,00
	2	3,00	3,00	6,00	
2	3	10,00	3,00	7,70	9,87
	4	17,70	6,00	2,17	
3	5	20,00	6,00	6,40	6,40

Por último, también hay que destacar que para resolver el problema presentado se tiene en cuenta las características siguientes:

- La línea de montaje es en serie y se procesa un único producto.
- El número de tareas a realizar en la línea y el tiempo de proceso sin deterioro de cada una de ellas es un dato. Una vez se inicia una tarea no se puede interrumpir, es indivisible.
- Se conocen las relaciones de precedencia que existen entre las distintas tareas. Cada tarea está disponible siempre y cuando ha finalizado su/s predecesora/s.
- Todas las estaciones son iguales, de modo que se puede asignar cualquier tarea a cualquier estación.
- El tiempo de ciclo se conoce y es el mismo para cada estación. Además, en cada estación la primera tarea puede no empezar al inicio del ciclo y puede haber tiempos ociosos entre las tareas.
- Es irrelevante para este problema si la estación está compuesta por un operador (humano o robot), cualquier tipo de máquina o mecanismo especializado.
- El tipo de deterioro que se considera es lineal. Es un dato y no tienen por qué tener todas las tareas la misma tasa de deterioro.

5. Resolución del problema

Tal y como se ha comentado, pese a que por lo general los métodos exactos garantizan una solución óptima, estos tienen como inconveniente que solo pueden utilizarse en casos con un número muy reducido de tareas, ya que para problemas que tienen un gran número de variables y de restricciones se hace inabordable en cuestión de tiempo de cálculo y recursos. Por este motivo, para el presente trabajo, se ha llevado a cabo el diseño de un modelo matemático y uno heurístico, con el objetivo de poder resolver tanto ejemplares pequeños como grandes (con un número más realista de tareas).

Entonces, para poder formalizar los dos procedimientos de resolución, hay que tener presente que se han utilizado como base los ejemplares del SALBP-1 ya definidos por *Otto et al.* (2013) [18]. Los datos que incluyen estas muestras son: el número de tareas, la parte constante de los tiempos de procesamiento de cada una de las tareas sin deterioro, el tiempo de ciclo máximo de cada estación y las relaciones de precedencia que existen entre las distintas tareas. El formato que presentan es el siguiente:

```
<number of tasks>
50

<cycle time>
1000

<order strength>
0.196


<task times>
1 73
2 125
3 172
4 58
5 139
6 91
7 166
8 76
9 210
10 116
11 72
12 203
13 141
14 95
15 256
```

```
<precedence relations>
1,33
2,8
2,9
4,11
4,38
5,10
5,30
6,7
7,33
8,13
9,12
9,14
9,15
9,17
11,12
11,16
12,33
13,18
14,19
14,21
15,23
15,30
15,32
```

Figura 5.1. Ejemplo de una parte de los datos que contiene un ejemplar de 50 tareas. [Fuente: propia]

Es importante tener en cuenta este formato porque los datos que contienen son los que se utilizarán para la resolución del problema. Además, no incluyen la tasa de deterioro y, por tanto, se deberá añadir a cada ejemplar.

5.1. Resolución con método exacto

En este apartado, para resolver el problema de ALBP-1 considerando el deterioro lineal de las tareas, se presenta el modelo matemático formulado para obtener una solución cuando el problema es simple. Para ello, como método exacto se utiliza la programación matemática, más concretamente, la Programación Lineal Entera Mixta (PLEM).

De nuevo destacar que, tal y como se ha dicho anteriormente, los métodos exactos garantizan una solución óptima (si es que existe), pero que por lo general, se utilizan en problemas con un número reducido de tareas, ya que cuando las dimensiones son elevadas el problema se vuelve *NP-hard* y se hace intratable por tiempo de cálculo y recursos.

Entonces, aunque existen muchos lenguajes para realizar la programación matemática, el modelo lineal que se propone se resuelve mediante el software IBM ILOG CPLEX, ya que es uno de los más eficientes y el que se ha utilizado a lo largo del máster y, por tanto, no se requiere de formación previa.

Por otra parte, para la programación de algunas de las restricciones del modelo y de las cotas, así como, de la interfaz que permita leer todos los ejemplares, se utiliza el editor *Visual Basic* que está incorporado en el Excel. Este entorno de programación funciona a través de Macros de Excel que permiten crear tareas automatizadas paso a paso y de forma sencilla. Se ha elegido esta herramienta por la familiarización del proyectista con este programa, el cual ha utilizado en numerosas ocasiones a lo largo de sus estudios.

A continuación, se va a presentar los pasos realizados para la implementación de este método. Los códigos programados para cada uno de ellos se encuentran en el Anexo B.1.

5.1.1. Formalización del modelo matemático

Datos

ct Tiempo de ciclo

N Conjunto de tareas $N = \{1, \dots, n\}$, donde n es el número de tareas

t_i Tiempo de procesamiento para realizar la tarea $i \in N$ sin deterioro

b_i Tasa de deterioro de la tarea $i \in N$

IP_i Conjunto de predecesores inmediatos de la tarea $i \in N$

P_i Conjunto de todos los predecesores de la tarea $i \in N \mid P_i = IP_i \cup \left(\bigcup_{h \in IP_i} P_h\right)$

$\bar{P}$ Conjunto de pares de tareas que no tienen relaciones de precedencia entre ellas
$\bar{P} = \{(h \in \{1, \dots, n-1\}, i \in \{h+1, \dots, n)\} \mid h \notin P_i \wedge i \notin P_h\}$

Cotas

lb^m (ub^m)	Límite inferior (superior) del número de estaciones de trabajo
M	Conjunto de estaciones que se pueden utilizar: $M = \{lb^m + 1, \dots, ub^m\}$
$e_i(l_i)$	Primera (última) estación a la que la tarea $i \in N$ se puede asignar
W_i	Conjunto de estaciones a las que se puede asignar la tarea $i \in N \mid W_i = \{e_i, \dots, l_i)$
DW	Conjunto de pares de tareas que se deben asignar a estaciones de trabajo diferentes: $DW = \{(h \in \{1, \dots, n-1\}, i \in \{h+1, \dots, n\}) \mid W_h \cap W_i = \emptyset\}$
lb_i^{st} (ub_i^{st})	Límite inferior (superior) en el tiempo de inicio de la tarea $i \in N$
lb_i^{av} (ub_i^{av})	Límite inferior (superior) en el tiempo en el que todos los predecesores de la tarea $i \in N \mid IP_i \neq \emptyset$ se han realizado
lb_i^{p} (ub_i^{p})	Límite inferior (superior) en el tiempo para realizar la tarea $i \in N$

Variables

$x_{ij} \in \{0,1\}$	1 si la tarea $i \in N$ está asignada a la estación $j \in W_i$, 0 en caso contrario
$y_j \in \{0,1\}$	1 si la estación de trabajo $j \in M$ se utiliza, 0 en caso contrario
$lb_i^{st} \leq st_i \leq ub_i^{st}$	Tiempo de inicio de la tarea. Para cada unidad de producto, el instante en el que la unidad llega a la línea se considera como tiempo 0
$lb_i^{av} \leq av_i \leq ub_i^{av}$	Tiempo en el que todos los predecesores de la tarea $i \in N \mid IP_i \neq \emptyset$ se han realizado
$lb_i^{p} \leq p_i \leq ub_i^{p}$	Tiempo de procesamiento para realizar la tarea $i \in N$
$w_{hi} \in \{0,1\}$	1 si la tarea h se realiza antes que la tarea i $((h,i) \in \bar{P} \backslash DW)$, 0 en caso contrario
$v_{hi} \in \{0,1\}$	1 si la tarea $h \in IP_i$ es la última predecesora de la tarea $i \in N \mid \lvert IP_i \rvert \geq 2$ a realizar, 0 en caso contrario

Función objetivo

$$[MIN] \sum_{j \in M} j \cdot y_j \qquad \text{(Ec. 5.1)}$$

Restricciones

$\sum_{j \in W_i} x_{ij} = 1$	$i \in N$	(Ec. 5.2)
$st_h + p_h \leq st_i$	$i \in N,\ h \in IP_i$	(Ec. 5.3)
$st_h + p_h \leq st_i + \left(ub_h^{st} + ub_h^{p} - lb_i^{st}\right) \cdot (1 - w_{hi})$	$(h,i) \in \bar{P} \backslash DW$	(Ec. 5.4)
$st_i + p_i \leq st_h + \left(ub_i^{st} + ub_i^{p} - lb_h^{st}\right) \cdot w_{hi}$	$(h,i) \in \bar{P} \backslash DW$	(Ec. 5.5)
$st_i \geq ct \cdot \left(\sum_{j \in W_i} (j-1) \cdot x_{ij}\right)$	$i \in N$	(Ec. 5.6)
$st_i + p_i \leq ct \cdot \left(\sum_{j \in W_i} j \cdot x_{ij}\right)$	$i \in N$	(Ec. 5.7)
$\sum_{h \in IP_i} v_{hi} = 1$	$i \in N \mid \lvert IP_i \rvert \geq 2$	(Ec. 5.8)
$av_i \geq st_h + p_h$	$i \in N \mid \lvert IP_i \rvert \geq 2,\ h \in IP_i$	(Ec. 5.9)
$av_i \leq st_h + p_h + \left(ub_i^{av} - lb_h^{st} - lb_h^{p}\right) \cdot (1 - v_{hi})$	$i \in N \mid \lvert IP_i \rvert \geq 2,\ h \in IP_i$	(Ec. 5.10)
$av_i = st_h + p_h$	$i \in N \mid \lvert IP_i \rvert = 1,\ h \in IP_i$	(Ec. 5.11)
$p_i = t_i + b_i \cdot st_i$	$i \in N \mid IP_i = \emptyset$	(Ec. 5.12)
$p_i = t_i + b_i \cdot (st_i - av_i)$	$i \in N \mid IP_i \neq \emptyset$	(Ec. 5.13)
$\sum_{i \in N \mid j \in W_i} x_{ij} \leq n \cdot y_j$	$j \in M$	(Ec. 5.14)
$y_k \leq y_j$	$j \in M \backslash \{ub^m\}, k \in \{j+1, \dots, ub^m\}$	(Ec. 5.15)

El modelo se puede explicar de la forma siguiente:

- La función objetivo (**Ec. 5.1**) consiste en minimizar el número de estaciones utilizadas dado un tiempo de ciclo concreto y un número de tareas determinado. Además, formulándola de este modo se logra tener una función objetivo equivalente y, en consecuencia, se rompen simetrías entre soluciones equivalentes.

- Con la **Ec. 5.2** se consigue que cada tarea se asigne únicamente a una estación de trabajo dentro de todo el conjunto de estaciones a las que se puede asignar.

- Las relaciones de precedencia de cada tarea se aseguran por medio de la **Ec. 5.3**. Esta restricción fija que el tiempo de inicio de cada tarea siempre sea como mínimo igual al instante en el que su tarea o tareas predecesoras inmediatas han finalizado

.

- Las **Ec. 5.4 y 5.5** garantizan que los pares de tareas que no tienen relaciones de precedencia entre ellas no se realicen simultáneamente en la misma estación. Aquí directamente ya no se tienen en cuenta los pares de tareas que se deben asignar a estaciones de trabajo diferentes.

- Las **Ec. 5.6 y 5.7** aseguran que cada tarea empiece y termine en la estación de trabajo asignada, siempre respetando el tiempo de ciclo máximo dado.

- Por medio de la **Ec. 5.8** se impone que para cada tarea solamente una de sus tareas predecesoras inmediatas sea la última en realizarse.

- Las **Ec. 5.9 y 5.10** fijan el tiempo en el que cada tarea ya está disponible para realizarse, una vez han finalizado todas sus predecesoras. Estas dos restricciones se aplican siempre y cuando el conjunto de sus predecesoras inmediatas sea igual o mayor que 2. Mientras que la **Ec. 5.11** se aplica cuando el conjunto de predecesoras inmediatas es 1. En caso que la tarea *h* sea la última predecesora de la tarea *i*, el tiempo en el que *i* estará disponible siempre será igual al instante en que finalice *h*.

- Por un lado, la **Ec. 5.12** establece que para cada tarea, que no tenga un conjunto de predecesoras inmediatas, el tiempo de procesamiento sea una función lineal de la tasa de deterioro de dicha tarea una vez se empieza a procesar. Por otro lado, para las tareas con un conjunto de predecesoras inmediatas diferente a vacío, se utiliza la **Ec. 5.13**. En esta última también se define el tiempo de procesamiento de cada tarea en función de su tasa de deterioro, pero esta vez teniendo en cuanta el deterioro sufrido desde que la tarea está disponible hasta el instante en el que se empieza a procesar.

- Finalmente, con la **Ec. 5.14** se logra que para cada estación, dentro del conjunto de estaciones que se pueden utilizar, el sumatorio de todas las tareas asignadas a dicha estación sea como máximo igual al número de tareas que se pueden asignar en total. La **Ec. 5.15** por su parte, se encarga de restringir que para cada estación utilizada dentro del rango de estaciones usables, sin incluir el número límite superior de estaciones de trabajo que se pueden emplear, pueda haber o no una estación siguiente a utilizar.

5.1.2. Cálculo de las cotas

Las cotas definidas, en el apartado anterior, se han incorporado con el objetivo de facilitar la resolución del modelo matemático y evaluar la calidad de las soluciones que se encuentren. Como se ha comentado, cada una de ellas se programa mediante el editor de *Visual Basic* de Excel por medio de los cálculos que se presentan a continuación.

Nota adicional:

IS$_i$ se refiere al conjunto de tareas sucesoras inmediatas de la tarea *i* ***ϵ*** *N*:

$$IS_i = \{h \in N \mid i \in IP_h\} \quad \text{(Ec. 5.16)}$$

S$_i$ es el conjunto de tareas sucesoras de la tarea *i* ***ϵ*** *N*:

$$S_i = \{h \in N \mid i \in P_h\} \quad \text{(Ec. 5.17)}$$

Cotas:

- **Cota *lb ^{m}*.** Para considerar el deterioro en el tiempo, sea $\Pi(X)$ la partición del conjunto de tareas *X*, tal que todas las tareas de un subconjunto de $\Pi(X)$ tengan las mismas predecesoras inmediatas (es decir, $\forall_{r\epsilon\Pi(X)}(\forall_{h\epsilon r}\,\forall_{i\in r}\, IP_h = IP_i)$) y dos tareas de subconjuntos diferentes no tengan las mismas predecesoras (es decir, $\forall_{r_1\epsilon\Pi(X)}\forall_{r_2\epsilon\Pi(X)\setminus\{r_1\}}(\forall_{h\epsilon r_1}\,\forall_{i\in r_2}\, IP_h \neq IP_i)$), una cota inferior del tiempo de proceso de todas las tareas de un conjunto $r \in \Pi(X)$, $lb^{\Delta}(r)$, es el tiempo que requiere procesarlas consecutivamente ordenadas crecientemente según el ratio t_i/b_i. Por tanto:

$$lb^m = \left\lceil \sum_{\Upsilon\in\Pi(N)} lb^{\Delta}(\Upsilon) \Big/ ct \right\rceil \quad \text{(Ec. 5.18)}$$

- **Cota *ub ^{m}*.** Es el número de estaciones de una solución heurística, donde:

$$ub^m = n \quad \text{(Ec. 5.19)}$$

- **Cota *e$_i$* (*i* ***ϵ*** *N*).** La fórmula tradicional del SALBP-1 es $\left\lceil \left(t_i + \sum_{h\in P_i} t_h \right) \Big/ ct \right\rceil$. Ahora bien, parece que esta cota puede mejorarse del modo siguiente, donde $\hat{e}_i = \max_{h\in IP_i} e_h$ para $i \in N \mid IP_i \neq \varnothing$:

$$\begin{cases} e_i^1 = 1 & \text{si } IP_i = \varnothing \\ e_i^1 = \hat{e}_i - 1 + \left\lceil \left(t_i + \sum_{h\in P_i \mid e_h^1 = \hat{e}_i} t_h \right) \Big/ ct \right\rceil & \text{si } IP_i \neq \varnothing \end{cases} \quad \text{(Ec. 5.20)}$$

Otra posible mejora es incluyendo cotas de los tiempos de deterioro:

$$e_i^2 = \left\lceil \left(t_i + \sum_{\Upsilon \in \Pi(P_i)} lb^{\Delta}(\Upsilon) \right) \Big/ ct \right\rceil \quad \text{(Ec. 5.21)}$$

Finalmente,

$$e_i = \max\left(e_i^1, e_i^2\right) \quad \text{(Ec. 5.22)}$$

- **Cota l_i ($i \in N$).** La fórmula tradicional de SALBP-1 es $ub^m + 1 - \left\lceil \left(t_i + \sum_{h \in S_i} t_h \right) \Big/ ct \right\rceil$.

  Análogamente a e_i, se puede mejorar de dos maneras, donde $\hat{l}_i = \min_{h \in IS_i} l_h$ para $i \in N \mid IS_i \neq \varnothing$:

$$\begin{cases} l_i^1 = ub^m & \text{si } IS_i = \varnothing \\ l_i^1 = \hat{l}_i + 1 - \left\lceil \left(t_i + \sum_{h \in S_i \mid l_h^1 = \hat{l}_i} t_h \right) \Big/ ct \right\rceil & \text{si } IS_i \neq \varnothing \end{cases} \quad \text{(Ec. 5.23)}$$

$$l_i^2 = ub^m + 1 - \left\lceil \left(t_i + \sum_{\Upsilon \in \Pi(S_i)} lb^{\Delta}(\Upsilon) \right) \Big/ ct \right\rceil \quad \text{(Ec. 5.24)}$$

Además, otra tercera mejora sería observando que el tiempo de proceso de una tarea no puede superar el tiempo de ciclo:

$$\text{Para } i \in N \mid IP_i = \varnothing: \; p_i = t_i + b_i \cdot st_i \leq ct \rightarrow st_i \leq \frac{ct - t_i}{b_i} \rightarrow l_i^3 = \left\lceil \frac{1}{b_i} \cdot \left(1 - \frac{t_i}{ct}\right) \right\rceil \quad \text{(Ec. 5.25)}$$

$$\text{Para } i \in N \mid IP_i \neq \varnothing: \; l_i^3 = \left(\max_{h \in IP_i} l_h^3 \right) + \left\lceil \frac{1}{b_i} \cdot \left(1 - \frac{t_i}{ct}\right) \right\rceil \quad \text{(Ec. 5.26)}$$

Finalmente,

$$l_i = \min\left(l_i^1, l_i^2, l_i^3\right) \quad \text{(Ec. 5.27)}$$

- **Cota lb^{av}_i ($i \in N \mid IP_i \neq \varnothing$).** Teniendo en cuenta que $\hat{e}_i = \max_{h \in IP_i} e_h$:

$$lb_i^{av} = ct \cdot (\hat{e}_i - 1) + \sum_{\Upsilon \in \Pi\left(\{h \in P_i : e_h = \hat{e}_i\}\right)} lb^{\Delta}(\Upsilon) \quad \text{(Ec. 5.28)}$$

- **Cota ub^{av}_i ($i \in N \mid IP_i \neq \varnothing$).**

$$ub_i^{av} = ct \cdot \left(\max_{h \in IP_i} l_h \right) \quad \text{(Ec. 5.29)}$$

- **Cota lb^{st}_i ($i \in N$).**

$$\text{Para } i \in N \mid IP_i = \varnothing \rightarrow lb_i^{st} = 0 \quad \text{(Ec. 5.30)}$$

$$\text{Para } i \in N \mid IP_i \neq \varnothing \rightarrow lb_i^{st} = max\left(lb_i^{av}, ct \cdot \left(e_i - 1\right)\right) \quad \text{(Ec. 5.31)}$$

- **Cota ub^{st}_i ($i \in N$).**

$$ub_i^{st} = ct \cdot l_i - t_i - \sum_{\Upsilon \in \Pi\left(\{h \in S_i \mid l_h = l_i\}\right)} lb^{\Delta}\left(\Upsilon\right) \quad \text{(Ec. 5.32)}$$

- **Cota lb^{p}_i ($i \in N$).**

$$\text{Para } i \in N \mid IP_i = \varnothing \rightarrow lb_i^{p} = t_i \quad \text{(Ec. 5.33)}$$

$$\text{Para } i \in N \mid IP_i \neq \varnothing \rightarrow lb_i^{p} = t_i + b_i \cdot \left(lb_i^{st} - lb_i^{av}\right) \quad \text{(Ec. 5.34)}$$

- **Cota ub^{p}_i ($i \in N$).**

$$\text{Para } i \in N \mid IP_i = \varnothing \rightarrow ub_i^{p} = t_i + b_i \cdot ub_i^{st} \quad \text{(Ec. 5.35)}$$

$$\text{Para } i \in N \mid IP_i \neq \varnothing \rightarrow ub_i^{p} = t_i + b_i \cdot \left(ub_i^{st} - lb_i^{av}\right) \quad \text{(Ec. 5.36)}$$

5.1.3. Interfaz entre los ejemplares y software CPLEX

En la Figura 5.1, se ha presentado cual es el formato que tienen los ejemplares que se van a utilizar en la resolución del ALBP-1. Ahora bien, este formato de datos no es el mismo que con el que trabaja el software IBM ILOG CPLEX.

Por este motivo, por medio del editor de *Visual Basic* de Excel, se ha creado un programa que básicamente lo que hace es leer de una carpeta de origen los ejemplares definidos (archivos "*.alb*"), acto seguido los transforma en la configuración deseada para que el software los pueda interpretar, y finalmente guarda estos datos generados en un nuevo archivo "*.dat*" (que es el tipo de archivo que lee el CPLEX) dentro de una carpeta de destino, la cual será la ruta que leerá directamente el modelo matemático creado en el software.

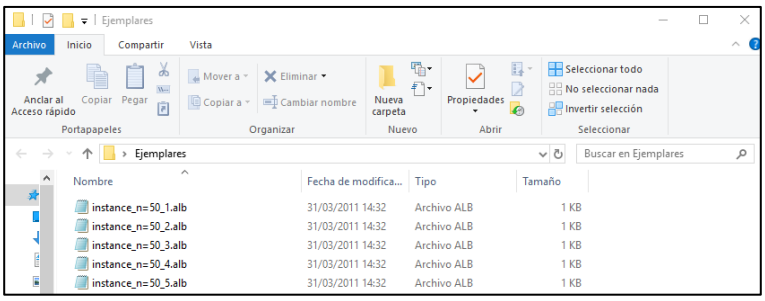


Figura 5.2. Carpeta de origen donde se guardan los ejemplares a utilizar para resolver el modelo. [Fuente: propia]

Figura 5.3. Software utilizado para transformar el diseño de los ejemplares. [Fuente: propia]

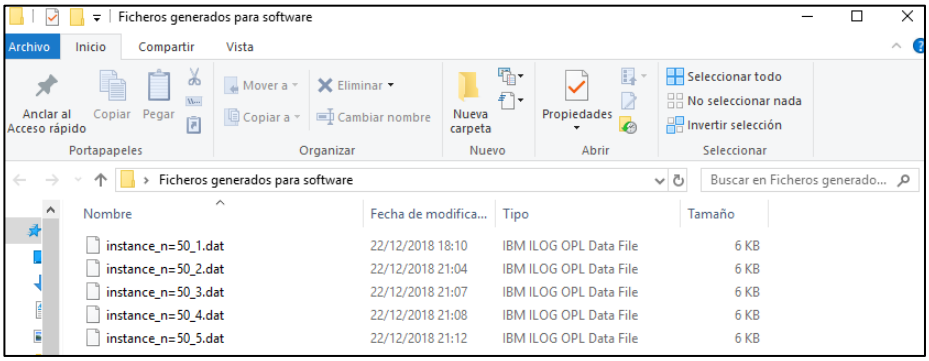


Figura 5.4. Carpeta de destino donde se almacenan los ejemplares transformados para que puedan ser leídos por el software. [Fuente: propia]

A continuación, para entender bien este problema de diseño que presentan los ejemplares, se muestra la transformación de los datos iniciales (los definidos en la Figura 5.1) en la configuración deseada y entendida por el software CPLEX para poder ejecutar el modelo presentado en el apartado 5.1.

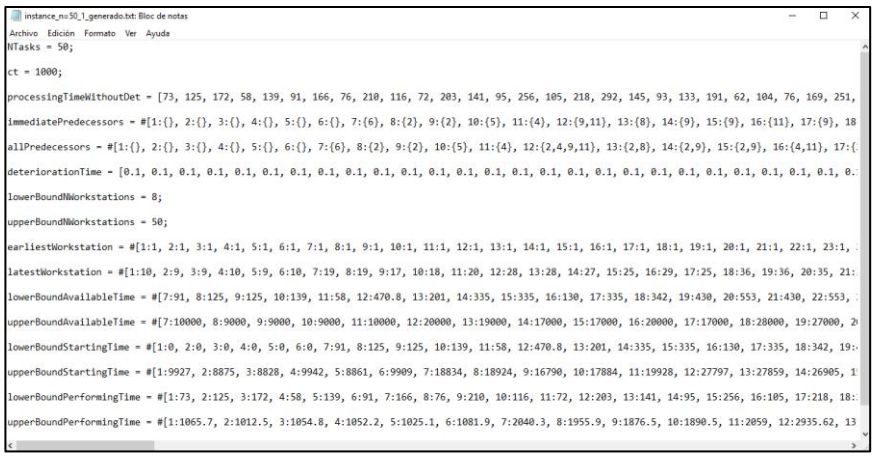


Figura 5.5. Nuevo archivo de texto generado con el formato necesario para que el software pueda leer el ejemplar. [Fuente: propia]

Finalmente, una vez se ha cambiado el diseño de todos los ejemplares que se van a resolver, ya se puede ejecutar el modelo matemático.

5.1.4. Ejecución del modelo en el software CPLEX

En primer lugar, hay que destacar que el software CPLEX trabaja ejecutando un proyecto que se compone de dos tipos distintos de archivos: "*archivo.mod*" y "*archivo.dat*". El primero contiene la programación del modelo matemático, es decir, la declaración de las variables, función objetivo y restricciones; mientras que en el segundo se definen los datos que va a emplear el "*archivo.mod*" para resolver el problema.

Ahora bien, en el caso del presente proyecto, el "*archivo.d*at" es cada uno de los distintos ejemplares que se quieren analizar, los cuales tal y como ya se ha especificado, se almacenan en la carpeta de destino, una vez se han transformado en la configuración correcta. Entonces, para poder leerlos todos juntos sin tener que interrumpir el programa cada vez que se tenga que utilizar uno diferente, se ha creado un programa intermedio dentro del mismo software que lo que hace es ejecutar el modelo matemático, definido en el "*archivo.mod*", con cada conjunto de datos que se encuentran en la carpeta.

5.2. Resolución con método aproximado

En este apartado se procede a explicar el procedimiento heurístico que se ha desarrollado para resolver el problema definido. Como se ha comentado, el objetivo de esto es encontrar una solución para los ejemplares que el modelo matemático no es capaz de resolver en un periodo de tiempo razonable.

Entonces, se ha elegido como método de resolución la optimización local. Esta se basa en generar soluciones vecinas a partir de una solución inicial dada y ver de este modo si se obtiene una solución vecina que mejore la actual, para substituirla y repetir de nuevo el proceso.

Por tanto, primero hay que diseñar como obtener una solución inicial que sirva de punto de partida para la optimización local. Para ello, se ha decidido utilizar una heurística y no un resultado aleatorio, con el fin de evitar una convergencia lenta al aplicarle luego los procesos de mejora. Luego, una vez obtenida, se implementará sobre ella la búsqueda local.

Todos los códigos programados se encuentran en el Anexo B.2 y se han adaptado al formato que tienen por defecto los ejemplares.

5.2.1. Generación de la solución inicial

Para generar la solución inicial se ha elegido los algoritmos *greedy*, ya que son con los que se ha trabajado en algunas asignaturas del máster y, además, son bastante rápidos y fáciles de implementar.

En primer lugar, hay que tener en cuenta que como el problema que se pretende resolver es un ALBP-1, se conoce el tiempo de ciclo máximo (TC). También se tiene como datos iniciales el número de tareas y el tiempo de procesamiento constante de cada una de ellas, así como, las relaciones de precedencia.

Así pues, el procedimiento diseñado se basa en que primero se define la regla de prioridad y a partir de esta se ordenan las tareas, obteniendo de este modo la lista de prioridad que se va a utilizar en el algoritmo. En caso de empate entre tareas, la regla que desempata es el orden numérico.

Una vez generada la lista, el siguiente paso es seguir su secuencia para ir temporizando las tareas en las estaciones, cumpliendo siempre con el TC. De manera que sólo se recorrerá una vez la lista de principio a fin y cada vez que se pueda asignar una tarea a una estación, se sustituirá en la lista su número por un cero, así hasta que se hayan sustituido todas, lo que querrá decir que ya han sido asignadas y que se ha obtenido la solución inicial.

Entonces, se empieza abriendo la primera estación y comprobando si en ella se puede colocar la tarea de la primera posición de la lista. Para que una tarea sea candidata, se debe cumplir que todas sus tareas predecesoras ya hayan sido asignadas previamente. En caso que se cumpla, se sustituirá esta tarea por un cero en la lista y se calculará su tiempo de procesamiento, aplicando la fórmula 4.1 ($p_i = t_i + b_i \cdot (st_i - av_i)$) y el procedimiento presentado al definir el problema en el apartado 4.2 (utilizado también en el modelo matemático). Si la tarea cabe en la estación en curso (al añadirla no se supera el TC), se asignará a esta y se calculará de nuevo el tiempo de la estación; sin embargo, si no cabe, se descarta. De manera que primero se intenta llenar todo lo posible la estación en curso, y cuando no cabe nada más, se cierra y se abre una nueva estación. Cada estación que se abra empezará por el tiempo de ciclo máximo del anterior, excepto la primera, que se iniciará en el instante “0”. Por ejemplo, si se abren dos estaciones y el TC=10 s, la primera empezará en el instante “0 s”, y la segunda lo hará a los “10 s”.

Por otro lado, en caso que la tarea no pueda ser asignada, esta no se pondrá a cero en la lista y automáticamente se pasará a la siguiente tarea de la secuencia, comprobando si está es una candidata o no. Este proceso se va repitiendo secuencialmente hasta llegar al final de la lista. Cada vez que se analiza una posición de la lista, si en las posiciones anteriores a esta todavía hay alguna tarea que no se ha podido asignar, primero se mira si se puede colocar en la estación y luego la tarea de la posición en curso.

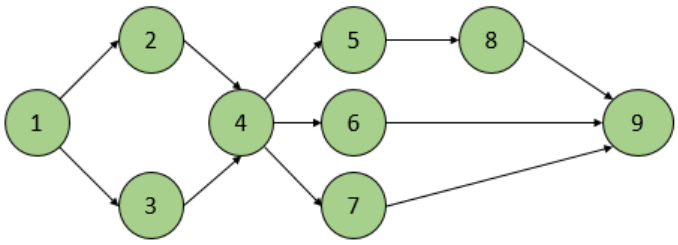
Finalmente, una vez programadas todas las tareas, se obtiene una solución inicial rápida con el número de estaciones necesarias basada en una regla heurística concreta, aunque no tiene por qué ser la óptima.

Se han definido 5 procedimientos heurísticos, mostrados en la Tabla 5.1, con el objetivo de analizar el impacto que tiene la tasa de deterioro en la resolución del problema, y es por este motivo, algunas reglas contemplan la tasa de deterioro y otras no. De este modo, se generarán 5 soluciones iniciales para cada ejemplar resuelto, que servirán de punto de partida para realizar la optimización local.

Tabla 5.1. Reglas de prioridad utilizadas para generar soluciones iniciales.

Regla de prioridad	Descripción	Formulación
H1	Ordenar por mayor tiempo de proceso constante de la tarea *i*	t_i
H2	Ordenar por mayor número de sucesoras inmediatas de la tarea *i*	IS_i
H3	Ordenar por mayor tiempo de proceso constante de la tarea *i* por su tasa de deterioro	$t_i \cdot b_i$
H4	Ordenar por mayor número de sucesoras inmediatas de la tarea *i* por su tasa de deterioro	$IS_i \cdot b_i$
H5	Ordenar por menor tiempo de proceso constante de la tarea *i* dividido entre su tasa de deterioro	$t_i \,/\, b_i$

Por último, para ejemplificar el proceso que se ha definido se resuelve el problema de *Jaeschke*, el cual consta de 9 tareas y sus relaciones de precedencia son las que se muestran en Figura 5.6. El tiempo de ciclo dado es de 14 segundos, la tasa de deterioro (b_i) de 0,1 para todas las tareas y la heurística que se utiliza es la H1, definida en la Tabla 5.1.



i	t_i
1	5
2	3
3	4
4	5
5	4
6	5
7	1
8	4
9	6

Figura 5.6. Grafo de precedencias del problema de Jaeschke. [Fuente: propia]

Así pues, siguiendo la regla H1, al ordenar las tareas por mayor tiempo de proceso constante la lista de prioridad que se genera es la siguiente:

9	1	4	6	3	5	8	2	7

Figura 5.7. Lista con la secuencia de tareas según la regla de prioridad H1. [Fuente: propia]

Una vez generada la lista, ya se puede empezar a asignar las tareas en las estaciones. Se empieza con la tarea 9 (primera posición), se observa que sus predecesoras todavía no han sido asignadas, y por tanto, se pasa a la siguiente tarea, es decir, a la tarea 1. En esta última, se vuelve a hacer de nuevo el proceso y se comprueba que si se puede colocar ya que no tiene predecesoras, por eso, es la primera asignada en la primera estación, se le da un valor de 0 a la lista y se pasa a la posición siguiente.

9	0	4	6	3	5	8	2	7

Figura 5.8. Secuencia de la lista ordenada de tareas al analizar la posición 3. [Fuente: propia]

Ahora en la posición 3, se mira primera si en las posiciones anteriores hay alguna tarea pendiente de asignar, y en caso que sí, se mira si se puede colocar en la estación. En este caso, está la tarea 9, la cual todavía no se puede asignar y entonces se pasa a mirar la de la posición en curso, la tarea 4. Esta también se descarta, ya que sus predecesoras son la tarea 2 y 3 y todavía no han sido colocadas. Luego, en la posición 4 pasa lo mismo que en la posición 3, no se puede asignar nada aún; y en la posición 5, se comprueba que la tarea 9, 5 y 6 no se pueden colocar, pero la tarea 3 sí. Acto seguido, hasta llegar a la posición 8 no se puede hacer nada más.

9	0	4	6	0	5	8	2	7

Figura 5.9. Secuencia de la lista ordenada de tareas al analizar la posición 8. [Fuente: propia]

En la posición 8, se comprueba si en las posiciones anteriores de la lista hay alguna tarea que se pueda asignar, que en este caso es negativo. En cambio, la tarea de la posición en curso, es decir, la tarea 2, sí que se asigna en la estación, ya que su predecesora es la tarea 1 y ya ha terminado, y además, su tiempo de proceso no supera el TC de la estación 1. A partir de este momento, se abre la estación 2, ya que en la anterior no cabe ninguna más. Este proceso se va repitiendo sucesivamente hasta colocar todas las tareas de la lista.

En la Tabla 5.2 se recoge la solución inicial obtenida aplicando la regla heurística H1. En ella se puede ver la distribución de las tareas y el número de estaciones necesarias, así como, el tiempo de procesamiento de cada una de ellas (p_i). También se observa que ninguna estación funciona a su tiempo de ciclo máximo, y por tanto, hay tiempos muertos entre una y otra. El procedimiento para obtener el tiempo de inicio (st_i) de la tarea y el momento en el que está disponible (av_i) es el mismo que el comentado en el apartado 4.2.

Tabla 5.2. Solución inicial obtenida para el problema de Jaeschke con TC = 14 s.

Nº estación	Tarea *i*	st_i (s)	av_i (s)	p_i (s)	Tiempo en la estación (s)	
1	1	0,00	0,00	5,00	5,00	12,40
	3	5,00	5,00	4,00	9,00	
	2	9,00	5,00	3,40	12,40	
2	4	14,00	12,40	5,16	19,16	25,66
	6	19,16	19,16	5,00	24,16	
	7	24,16	19,16	1,50	25,66	
3	5	28,00	19,16	4,88	32,88	36,88
	8	32,88	32,88	4,00	36,88	
4	9	42,00	36,88	6,51	48,51	48,51

Llegados a este punto, una vez desarrollado el algoritmo *greedy* para obtener la solución inicial, se procede a aplicarle el método de búsqueda local para encontrar una mejor solución.

5.2.2. Generación de las soluciones vecinas

Para llevar a cabo la optimización local, se ha diseñado un método que permite generar soluciones vecinas por medio del software de *Visual Basic* de Excel.

Es importante destacar que existen distintas formas de obtener el vecindario de una solución inicial de partida. En este trabajo, teniendo en cuenta la programación de las tareas en las distintas estaciones de la solución inicial, se ha probado generar las soluciones vecinas cediendo tareas entre estaciones, para ver si alguna estación se puede quedar libre de tareas y obtener así una solución con un menor número de estaciones necesarias.

Esto significa que una "*tarea a"* se probará de ceder a una "*estación m"* donde exista tiempo muerto suficiente, ya que así se podrá asegurar que hay la posibilidad de que quepa porque la estación no está llena. Además, se define que puede cederse en cualquier posición de la estación, es decir, si en una estación hay 2 tareas, se probará si la "*tarea a"* cabe delante de la tarea 1, delante de la 2 y detrás de la 2, siempre que se respeten las relaciones de precedencia.

En definitiva, hay que tener en cuenta que para que una "*tarea a"* se pueda ceder a otra "*estación m"* se debe cumplir con las condiciones siguientes:

1) Todas las predecesoras de la "***tarea a****"*, se han realizado antes de la "***posición b***" que va a ocupar en la "***estación m***"

2) Todas las sucesoras de la "***tarea a***", se encuentran como mínimo después de la "***posición b***" que va a ocupar en la "***estación m***"

3) Al ceder la "***tarea a***" en la "***posición b***" de la "***estación m***", no se puede superar el TC máximo permitido de dicha estación

Así pues, para cada una de las tareas existentes se comprueba, analizando si se cumplen las condiciones definidas, si se puede ceder en cualquier posición de cualquier estación que sea distinta a la que se encuentra. Entonces, si se cumplen las 3 condiciones se cederá la tarea en la estación y se calcularán de nuevo todos los tiempos de cada una de las tareas y estaciones. En caso contrario, se descarta la cesión y se pasa a la tarea siguiente.

A continuación, con el fin de entenderlo mejor, se va a ilustrar este proceso siguiendo con el ejemplo de *Jaeschk*e. Para ello, se comprueba sobre la secuencia obtenida en la solución inicial (Tabla 5.2), si hay alguna tarea que se pueda ceder a otra estación.

Estación 1	1	3	2
Estación 2	4	6	7
Estación 3	5	8	
Estación 4	9		

Figura 5.10. Solución inicial obtenida para el problema de Jaeschke aplicando el algoritmo de greedy diseñado. [Fuente: propia]

Sobre la solución de la Tabla 5.2, el algoritmo encuentra distintas opciones de cesión de tareas. Un ejemplo sería la cesión de la tarea 6 (de la estación 2), a la estación 4 por delante de la tarea 9:

Estación 1	1	3	2
Estación 2	4	6	7
Estación 3	5	8	
Estación 4	9		

Figura 5.11. Ejemplo de la generación de una solución vecina de la solución actual. [Fuente: propia]

Esto es posible porque se cumplen las tres condiciones comentadas: todas las predecesoras de la tarea 6 se encuentran antes que la tarea 9 de la estación 4, y su sucesora (que es solo la tarea 9) se halla detrás. Además, recalculando el tiempo de ciclo de todas las tareas y estaciones, se observa que no se supera el TC máximo permitido (14 segundos):

Tabla 5.3. Una solución vecina generada para el problema de Jaeschke con TC = 14 s.

Nº estación	Tarea *i*	st_i (s)	av_i (s)	p_i (s)	Tiempo en la estación (s)	
1	1	0,00	0,00	5,00	5,00	12,40
	3	5,00	5,00	4,00	9,00	
	2	9,00	5,00	3,40	12,40	
2	4	14,00	12,40	5,16	19,16	20,16
	7	19,16	19,16	1	20,16	
3	5	28,00	19,16	4,88	32,88	36,88
	8	32,88	32,88	4,00	36,88	
4	6	42,00	19,16	7,28	49,28	55,28
	9	49,28	49,28	6	55,28	

Por último, ahora habrá que ver, si esta solución vecina de la Tabla 5.3 mejora la solución inicial de la Tabla 5.2. Esto se procede a explicar en el próximo apartado.

5.2.3. Selección de la mejor solución vecina

Una vez desarrollado el algoritmo para generar las soluciones vecinas a partir de la solución inicial, hay que tener en cuenta que el método de optimización local presenta dos variantes. Estas se diferencian por el momento en el que se selecciona que una solución vecina mejora la inicial:

- **Búsqueda exhaustiva**: se generan todas las soluciones vecinas de la solución actual considerada y se evalúan. Después, de todas las soluciones vecinas que se obtienen, se elige la mejor de ellas y si esta resulta ser mejor que la actual pasará a ser la nueva solución inicial de la que se parte. Sobre esta, se volverá a repetir el proceso buscando todas las soluciones vecinas y comprobando si se logra alguna mejor. La búsqueda finaliza cuando ninguna de las soluciones vecinas que se encuentran mejora la solución actual, lo que quiere decir que se ha llegado al óptimo local.

- **Búsqueda no exhaustiva:** se comienza generando las soluciones vecinas de la solución actual y cuando se encuentre alguna que la mejore pasa a considerarse la nueva solución actual, sin la necesidad de haber buscado primero todas las soluciones vecinas posibles. Entonces, sobre la nueva encontrada, se vuelve a buscar de nuevo sus vecinas y así sucesivamente, hasta que no se encuentra ninguna solución que mejore la que se tenga en curso porque ya se ha alcanzado el óptimo local.

Entre estas dos opciones, se ha elegido la búsqueda no exhaustiva. Por tanto, para su implementación, utilizando también el *Visual Basic* de Excel, se ha programado que en el momento en el que se encuentre una solución vecina que mejore la inicial, esta pase a

considerarse como la nueva solución actual y sobre ella, se vuelva a generar su vecindario. De tal manera que se haga este proceso sucesivamente hasta que no se obtenga ninguna solución vecina que mejore la que se tenga.

Entonces, se considerará que el vecino generado es mejor que la solución actual si tiene un menor número de estaciones necesarias, ya que el objetivo del ALBP-1 es lograr minimizarlas. Por otra parte, en caso de que tengan el mismo número de estaciones, se desempatará utilizando el criterio de que gana la que tenga el mayor sumatorio de los cuadrados de la carga de tiempo de cada estación. De modo que si la solución vecina tiene una carga mayor, esta pasará a ser la nueva solución actual; en cambio, si la solución inicial es la que gana, se le generará otra nueva vecina y se volverá a evaluar. Esto se puede expresar de la forma siguiente (donde *m* es el número de estaciones, y W_j la carga de tiempo de cada estación *j*):

$$\sum_{j=1}^{m} W_j{}^2 \qquad \text{(Ec. 5.37)}$$

Gracias a este criterio de desempate, se tiende a soluciones más desequilibradas, es decir, habrá estaciones de la solución con baja carga de tiempo y otras con elevada. De este modo, se ayuda a la optimización a generar más vecinas para ver si alguna de ellas puede lograr reducir el número de estaciones, ya que en futuras iteraciones alguna de las estaciones con baja carga podría quedarse vacía.

Ahora, siguiendo con el ejemplo de *Jaeschke*, se elabora una tabla resumen para decidir si la solución vecina generada en la Tabla 5.3 es mejor o no que la solución inicial de la Tabla 5.2:

Tabla 5.4. Comparativa solución Inicial vs. solución vecina obtenida para el problema de Jaeschke.

Solución inicial	
Estaciones	**Tiempo estación**
1	12,40
2	25,66
3	36,88
4	48,51
$\sum_{j=1}^{m} W_j^2$	4.525,55

Solución vecina	
Estaciones	**Tiempo estación**
1	12,40
2	20,16
3	36,88
4	55,28
$\sum_{j=1}^{m} W_j^2$	4.976,20

Finalmente, se puede ver que en ambos casos se obtiene el mismo número de estaciones (m = 4) y entonces, hay que desempatar aplicando el criterio de la suma de los cuadrados de la carga de cada estación. El resultado es que la solución vecina tiene una carga mayor, y por tanto, se considera que mejora a la inicial y pasa a ser la nueva solución actual, sobre la cual se generaran soluciones vecinas y se repetirá el proceso hasta no encontrar ninguna que mejore.

6. Experiencia computacional

6.1. Ejemplares utilizados

Una vez definidos los dos procedimientos de resolución, con el objetivo de evaluar su funcionamiento, se ha realizado una experiencia computacional por medio de un conjunto de datos, que como ya se ha comentado se han descargado de forma online y gratuita [18]. Estos ejemplares han sido definidos por *Otto et al.* (2013) para utilizarlos en la resolución de los SALBP-1. Entonces, dado que estos se han empleado en muchos estudios sobre el equilibrado de líneas para probar y comparar los procedimientos, se ha decidido usarlos.

En primer lugar, es importante destacar que hay una gran cantidad de ejemplares disponibles y que estos se clasifican en los grupos que se muestran a continuación:

- Conjunto de ejemplares pequeños, con 20 tareas
- Conjunto de ejemplares medianos, con 50 tareas
- Conjunto de ejemplares grandes, con 100 tareas
- Conjunto de ejemplares muy grandes, con 1.000 tareas

Ahora bien, conforme el artículo que explica cómo se han generado estos conjuntos de datos [19], para probar métodos de soluciones exactas se recomienda utilizar el grupo de datos medio (con n=50 tareas). En cambio, para la resolución de heurísticas se comenta que el conjunto grande es el más adecuado, es decir, el de 100 tareas. Por tanto, ante estas recomendaciones, para la resolución del problema se aplica el conjunto de datos sugerido para cada tipo de procedimiento diseñado.

Por otra parte, también hay que tener en cuenta que dentro de cada conjunto de datos se dividen los ejemplares en 6 categorías distintas, donde la diferencia entre ellas es la medida de precisión estadística estimada (calculada en el estudio de *Otto et al.* (2013)) que hay de encontrar soluciones no óptimas en el espacio de soluciones. De manera que la probabilidad estimada de que una solución sea óptima en cada uno de los grupos es la siguiente:

- ***Trivial:*** 100%
- ***Less tricky:*** > 95%
- ***Tricky:*** 5% - 50%
- ***Very tricky:*** 0,5% - 5%
- ***Extremly tricky:*** < 0.5%
- ***Open:*** no se conoce la solución óptima

Entonces, de estos 6 grupos se ha descartado el *Trivial* por su sencillez, ya que para evaluar el funcionamiento de los procedimientos interesa que haya alguna dificultad a la hora de obtener una resolución.

Además, dentro de cada categoría hay tres tipos de fuerza de orden (*OS, Order strength*), la cual indica el nivel de relaciones de precedencia que hay entre las tareas del ejemplar. De modo que los ejemplares con **"*OS* baja"** tendrán una orden de 0,2, los de **"*OS* media"** de 0,6 y los de "**alta *OS*"** de 0,9. Una orden de 0,2 significa que hay pocas relaciones de precedencia entre las tareas, mientras que con 0,9 habrá fuertes relaciones entre ellas. Para entender bien cuál es la definición de *OS*, se muestra a continuación su fórmula, donde *n* es el número de tareas y E^T el número de relaciones de precedencias totales del grafo:

$$OS = \frac{2 \cdot |E^T|}{n \cdot (n-1)} \qquad \text{(Ec. 6.1)}$$

Así pues, con el objetivo de testear un número razonable de muestras que permitan obtener buenas conclusiones y teniendo en cuenta las consideraciones comentadas, se ha decidido seleccionar de forma aleatoria 16 ejemplares de cada una de las 5 categorías, donde entre los seleccionados estén presentes los 3 tipos de fuera de orden (*OS*). Este criterio se aplicará tanto para el conjunto de datos con n=50 tareas como para el de n=100 tareas.

Por tanto, se analizaran un total de 80 ejemplares para el método exacto y otros 80 para el método heurístico, los cuales se recogen en la Tabla 6.1 y la Tabla 6.2, respectivamente. Para cada ejemplar, se puede ver: su categoría, nombre, *OS*, el menor tiempo de proceso constante de la tarea más pequeña dividido entre el TC (Tmin/TC), el mayor tiempo de proceso constante de la tarea más grande dividido entre el TC (Tmax/TC), y Tsum/c.

6.1.1. Ejemplares para resolución del modelo matemático

En este caso, se ha establecido un límite de tiempo de 1h para encontrar la solución óptima en cada una de las instancias. Se considera que es un valor suficiente para obtener resultados a partir de los cuales extraer buenas conclusiones.

Es importante destacar que aquí cada ejemplar contiene 50 tareas y tiene un tiempo de ciclo (TC) de 1.000 segundos.

Tabla 6.1. Conjunto de datos con n=50 tareas y TC=1.000 s para resolver el método exacto.

Categoría	Nombre	"*OS*"	Tmin/TC	Tmax/TC	Tsum/TC
Less tricky	inst_n=50_1	0,2	0,027	0,292	7,276
	inst_n=50_2	0,2	0,022	0,329	5,585
	inst_n=50_3	0,2	0,020	0,349	7,491
	inst_n=50_4	0,2	0,033	0,303	6,703
	inst_n=50_5	0,2	0,020	0,310	6,778
	inst_n=50_76	0,6	0,029	0,327	6,630
	inst_n=50_80	0,6	0,023	0,379	6,578
	inst_n=50_81	0,6	0,024	0,326	6,357
	inst_n=50_82	0,6	0,022	0,381	5,750
	inst_n=50_84	0,6	0,025	0,354	6,752
	inst_n=50_85	0,6	0,026	0,386	7,184
	inst_n=50_452	0,9	0,030	0,338	7,163
	inst_n=50_453	0,9	0,021	0,350	6,348
	inst_n=50_454	0,9	0,023	0,396	6,986
	inst_n=50_455	0,9	0,021	0,304	5,636
	inst_n=50_456	0,9	0,023	0,367	7,209
Tricky	inst_n=50_7	0,2	0,020	0,333	6,838
	inst_n=50_22	0,2	0,024	0,324	6,826
	inst_n=50_25	0,2	0,021	0,306	5,908
	inst_n=50_55	0,2	0,026	0,571	12,227
	inst_n=50_57	0,2	0,035	0,527	12,359
	inst_n=50_58	0,2	0,026	0,578	10,529
	inst_n=50_77	0,6	0,028	0,299	6,830
	inst_n=50_78	0,6	0,025	0,323	6,872
	inst_n=50_83	0,6	0,028	0,381	7,805
	inst_n=50_91	0,6	0,025	0,381	6,805
	inst_n=50_94	0,6	0,026	0,336	6,826
	inst_n=50_451	0,9	0,039	0,341	7,563
	inst_n=50_459	0,9	0,021	0,360	6,672
	inst_n=50_461	0,9	0,026	0,300	5,962
	inst_n=50_462	0,9	0,026	0,340	6,842
	inst_n=50_464	0,9	0,027	0,328	5,843
Very tricky	inst_n=50_12	0,2	0,021	0,265	5,959
	inst_n=50_54	0,2	0,024	0,504	10,685
	inst_n=50_56	0,2	0,02	0,569	10,724
	inst_n=50_59	0,2	0,031	0,547	10,734
	inst_n=50_62	0,2	0,025	0,612	12,534
	inst_n=50_71	0,2	0,05	0,564	12,598
	inst_n=50_92	0,6	0,021	0,299	6,874
	inst_n=50_109	0,6	0,232	0,734	24,732
	inst_n=50_123	0,6	0,257	0,739	25,456
	inst_n=50_127	0,6	0,037	0,586	13,276
	inst_n=50_145	0,6	0,023	0,64	9,753
	inst_n=50_458	0,9	0,022	0,272	6,897
	inst_n=50_470	0,9	0,022	0,325	7,668
	inst_n=50_471	0,9	0,023	0,309	6,806
	inst_n=50_502	0,9	0,032	0,563	9,511
	inst_n=50_508	0,9	0,028	0,633	12,368

Extremly tricky	inst_n=50_26	0,2	0,147	0,72	24,218
	inst_n=50_27	0,2	0,265	0,759	26,048
	inst_n=50_29	0,2	0,148	0,74	24,61
	inst_n=50_34	0,2	0,226	0,79	25,782
	inst_n=50_35	0,2	0,341	0,735	26,393
	inst_n=50_36	0,2	0,18	0,867	25,579
	inst_n=50_79	0,6	0,025	0,314	7,925
	inst_n=50_90	0,6	0,031	0,338	6,961
	inst_n=50_97	0,6	0,029	0,398	6,91
	inst_n=50_107	0,6	0,121	0,815	25,423
	inst_n=50_108	0,6	0,274	0,684	25,381
	inst_n=50_475	0,9	0,026	0,348	5,917
	inst_n=50_479	0,9	0,259	0,812	24,514
	inst_n=50_480	0,9	0,341	0,811	26,634
	inst_n=50_490	0,9	0,215	0,672	24,613
	inst_n=50_495	0,9	0,336	0,82	26,763
Open	inst_n=50_28	0,2	0,216	0,812	25,173
	inst_n=50_30	0,2	0,241	0,783	24,972
	inst_n=50_31	0,2	0,31	0,778	24,561
	inst_n=50_43	0,2	0,269	0,684	23,988
	inst_n=50_48	0,2	0,284	0,754	25,008
	inst_n=50_102	0,6	0,295	0,824	26,092
	inst_n=50_105	0,6	0,14	0,869	22,49
	inst_n=50_106	0,6	0,239	0,734	25,163
	inst_n=50_116	0,6	0,187	0,728	25,56
	inst_n=50_121	0,6	0,282	0,738	26,006
	inst_n=50_122	0,6	0,294	0,812	25,812
	inst_n=50_476	0,9	0,257	0,757	24,051
	inst_n=50_477	0,9	0,291	0,731	24,402
	inst_n=50_478	0,9	0,234	0,79	25,291
	inst_n=50_481	0,9	0,271	0,751	24,027
	inst_n=50_482	0,9	0,197	0,676	23,71

6.1.2. Ejemplares para resolución del modelo heurístico

Para este caso no hay tiempo límite definido, a diferencia del modelo matemático, ya que se considera que los ejemplares se van a resolver en un tiempo suficientemente rápido. De manera que, la resolución del ejemplar se para cuando se llega al óptimo local.

Cada ejemplar contiene 100 tareas y tiene un tiempo de ciclo (TC) de 1.000 segundos.

Tabla 6.2. Conjunto de datos con n=100 tareas y TC=1.000 s para resolver el método heurístico.

Categoría	Nombre	"*OS*"	Tmin/TC	Tmax/TC	Tsum/TC
Less tricky	inst_n=100_30	0,2	0,026	0,292	14,284
	inst_n=100_31	0,2	0,023	0,374	13,113
	inst_n=100_32	0,2	0,022	0,466	13,425
	inst_n=100_33	0,2	0,021	0,417	14,184
	inst_n=100_34	0,2	0,021	0,340	14,488
	inst_n=100_37	0,2	0,028	0,350	13,310
	inst_n=100_102	0,6	0,020	0,301	13,014
	inst_n=100_103	0,6	0,022	0,396	13,148
	inst_n=100_105	0,6	0,021	0,307	12,383
	inst_n=100_106	0,6	0,024	0,352	13,043
	inst_n=100_109	0,6	0,022	0,347	14,226
	inst_n=100_480	0,9	0,021	0,312	13,803
	inst_n=100_482	0,9	0,021	0,341	13,832
	inst_n=100_483	0,9	0,023	0,384	12,977
	inst_n=100_485	0,9	0,027	0,369	14,837
	inst_n=100_486	0,9	0,023	0,379	14,041
Tricky	inst_n=100_167	0,2	0,020	0,581	21,062
	inst_n=100_176	0,2	0,023	0,350	12,684
	inst_n=100_177	0,2	0,021	0,399	13,679
	inst_n=100_193	0,2	0,021	0,469	14,645
	inst_n=100_194	0,2	0,022	0,351	14,559
	inst_n=100_253	0,6	0,029	0,290	13,685
	inst_n=100_254	0,6	0,023	0,371	13,642
	inst_n=100_267	0,6	0,021	0,410	12,700
	inst_n=100_270	0,6	0,029	0,393	12,645
	inst_n=100_272	0,6	0,021	0,330	13,551
	inst_n=100_475	0,9	0,022	0,571	22,186
	inst_n=100_477	0,9	0,022	0,331	13,492
	inst_n=100_478	0,9	0,023	0,272	13,494
	inst_n=100_484	0,9	0,025	0,292	13,405
	inst_n=100_494	0,9	0,024	0,327	13,546
	inst_n=100_497	0,9	0,023	0,419	12,504
Very tricky	inst_n=100_5	0,2	0,022	0,565	21,400
	inst_n=100_7	0,2	0,022	0,642	25,046
	inst_n=100_23	0,2	0,020	0,605	23,305
	inst_n=100_114	0,6	0,023	0,338	12,830
	inst_n=100_116	0,6	0,028	0,421	15,624
	inst_n=100_163	0,2	0,026	0,614	24,072
	inst_n=100_164	0,2	0,021	0,582	22,308
	inst_n=100_169	0,2	0,023	0,601	20,478
	inst_n=100_232	0,6	0,021	0,597	21,176
	inst_n=100_398	0,6	0,026	0,618	23,987
	inst_n=100_401	0,6	0,022	0,400	14,647
	inst_n=100_468	0,9	0,028	0,598	23,182
	inst_n=100_469	0,9	0,024	0,634	20,389
	inst_n=100_470	0,9	0,028	0,616	23,514
	inst_n=100_471	0,9	0,021	0,589	24,123
	inst_n=100_499	0,9	0,022	0,393	13,565

Extremly tricky	inst_n=100_22	0,2	0,022	0,571	23,870
	inst_n=100_25	0,2	0,029	0,607	21,640
	inst_n=100_26	0,2	0,025	0,287	13,912
	inst_n=100_27	0,2	0,022	0,400	12,929
	inst_n=100_28	0,2	0,021	0,340	13,881
	inst_n=100_97	0,6	0,024	0,625	21,519
	inst_n=100_98	0,6	0,022	0,553	21,260
	inst_n=100_100	0,6	0,020	0,571	24,598
	inst_n=100_101	0,6	0,030	0,352	14,834
	inst_n=100_244	0,6	0,022	0,577	20,336
	inst_n=100_246	0,6	0,031	0,568	25,359
	inst_n=100_472	0,9	0,024	0,580	21,973
	inst_n=100_473	0,9	0,023	0,611	26,773
	inst_n=100_481	0,9	0,022	0,343	14,609
	inst_n=100_489	0,9	0,020	0,353	12,840
	inst_n=100_492	0,9	0,021	0,301	13,701
Open	inst_n=100_57	0,2	0,168	0,788	50,006
	inst_n=100_60	0,2	0,135	0,761	50,084
	inst_n=100_175	0,2	0,022	0,593	25,915
	inst_n=100_208	0,2	0,174	0,806	50,761
	inst_n=100_214	0,2	0,165	0,792	49,710
	inst_n=100_277	0,6	0,26	0,793	50,718
	inst_n=100_278	0,6	0,244	0,863	50,908
	inst_n=100_279	0,6	0,216	0,901	49,826
	inst_n=100_290	0,6	0,195	0,872	49,930
	inst_n=100_397	0,6	0,023	0,582	24,869
	inst_n=100_429	0,6	0,233	0,918	51,365
	inst_n=100_502	0,9	0,223	0,775	51,657
	inst_n=100_504	0,9	0,142	0,723	50,018
	inst_n=100_505	0,9	0,228	0,738	49,907
	inst_n=100_523	0,9	0,194	0,758	48,309
	inst_n=100_525	0,9	0,205	0,768	49,822

6.2. Escenarios

Los ejemplares descargados no tienen tasa de deterioro, y por tanto, se ha tenido que añadir a cada uno de ellos. Entonces, dado que es un dato con el que se puede jugar y con el objetivo de analizar cómo afecta el deterioro de las tareas, se han definido 4 escenarios con una tasa de deterioro (b_i) distinta en cada uno de ellos:

- **Escenario 1:** b = 0 en todas las tareas (sin deterioro)
- **Escenario 2:** b = 0,1 en todas las tareas
- **Escenario 3:** b = 0,2 en todas las tareas
- **Escenario 4:** b = nº aleatorio en cada tarea comprendido entre el 0,01 y el 0,2

Por un lado, el motivo por el cual se han elegido los valores de b=0,1 y b=0,2 es porque son los más utilizados en la literatura existente (analizada en la Tabla 4.1) para el problema estudiado.

Por otro lado, también se ha planteado el escenario de b=0 para obtener resultados sin considerar el tiempo de deterioro en las tareas y poder compararlos con los otros 3 escenarios que sí lo consideran. Además, el cuarto escenario considera deterioros diferentes para las tareas.

En conclusión, la experiencia computacional se hará combinando los 4 escenarios con los ejemplares definidos en la Tabla 6.1 y Tabla 6.2, dando un total de 2.080 ejecuciones: 320 para el procedimiento matemático y 1.760 para el heurístico, tal y como se recoge en la Tabla 6.3. En el modelo heurístico se realiza una experiencia mayor para poder probar y comparar el modelo con distintas reglas heurísticas (Tabla 5.1), y además, también se resuelven los ejemplares de 50 tareas para poder comparar los resultados obtenidos con el modelo matemático.

Tabla 6.3. Nº ejemplares resueltos en la experiencia computacional por escenario y procedimiento.

		E1: b =0	**E2: b =0,1**	**E3: b =0,2**	**E4: b =aleatoria**
Modelo matemático (n= 50 tareas)		80	80	80	80
Modelo heurístico (n=50 tareas)	H1	80	80	80	80
	H2	80	80	80	80
	H3	-	-	-	80
	H4	-	-	-	80
	H5	-	-	-	80
Modelo heurístico (n=100 tareas)	H1	80	80	80	80
	H2	80	80	80	80
	H3	-	-	-	80
	H4	-	-	-	80
	H5	-	-	-	80

Por último, es importante destacar que las reglas heurísticas H1 y H2, no tienen en cuenta el deterioro a la hora de generar la lista ordenada de tareas a utilizar en el *greedy*. Entonces, en los escenarios 2 y 3, la tasa de deterioro definida sólo influirá a la hora de calcular el tiempo de proceso total de cada tarea.

En cuanto a las reglas H3, H4 y H5, como estas sí que tienen en cuenta el deterioro, se descarta directamente el escenario 1 y, además, solo se utilizarán en el escenario 4. En los escenarios 2 y 3 no tiene sentido porque estos tienen la misma tasa de deterioro en todas las tareas, y por tanto, no afecta como criterio de ordenación.

6.3. Análisis de los resultados

El problema que se estudia se trata de una ALBP-1, donde se considera que las tareas están disponibles una vez terminan sus predecesoras. También tiene en cuenta el deterioro lineal y no siempre tienen porque tener todas las tareas la misma tasa de deterioro.

Entonces teniendo en cuenta estas especificaciones, de los 8 estudios del ALBP con deterioro encontrados en la literatura (mostrados en la Tabla 4.1) sólo se podría seleccionar el de *Noushabadi et al.* (2011) para hacer una comparativa con los resultados obtenidos en el presente proyecto. Esto es debido a que este es con el único que se dan las mismas características del problema estudiado en este trabajo.

Ahora bien, este estudio también se ha descartado, ya que al analizar bien su artículo se han descubierto errores en la propuesta de resolución. Principalmente, como se ha comentado en el apartado 4.1 del estado del arte, es porque no tiene en cuenta los tiempos ociosos de las estaciones al calcular el deterioro, y por tanto, esto anula la validez de sus resultados.

6.3.1. Método exacto (modelo matemático)

Por medio del software IBM ILOG CPLEX, aplicando el modelo matemático se ha resuelto cada uno de los ejemplares de 50 tareas definidos en la Tabla 6.1. El detalle completo de los resultados obtenidos se puede encontrar en el Anexo A.1.

En primer lugar, antes de empezar con el análisis, hay que tener en cuenta los datos que devuelve CPLEX al resolver un ejemplar:

- ***GetCplexStatus***: devuelve un valor numérico que puede indicar lo siguiente:

  Tabla 6.4. Significado de los tipos de GetCplexStatus obtenidos en las soluciones.

Valor	Significado
1	Solución óptima demostrada por el propio Cplex
3	El ejemplar se considera infactible, no existe solución
11	Se detuvo la resolución por superar el tiempo límite, por tanto, se obtiene una solución sin optimalidad demostrada

  Es importante destacar que se puede dar el caso de obtener una solución que sea infactible cuando no se satisfacen todas las restricciones del problema.

- ***GetObjValue:*** devuelve el valor objetivo de la solución encontrada.
- ***GetBestObjValue*****:** devuelve el valor de función objetivo mínimo de todos los nodos no explorados restantes. Cuando un problema se ha resuelto de manera óptima, este valor coincide con el valor obtenido en *GetObjValue*.

Además de estos datos, para cada ejemplar también se obtiene el tiempo de ejecución y el número de estaciones totales requeridas.

A continuación, con el objetivo de evaluar el porcentaje de ejemplares que se han podido resolver en cada escenario, se ha elaborado la Tabla 6.5. La columna que indica que los ejemplares se han detenido quiere decir que han agotado el tiempo límite de resolución establecido (en este caso de 1h) y, por tanto, se ha parado el modelo y este ha devuelto el mejor resultado encontrado hasta ese momento (si es que ha encontrado alguno).

Tabla 6.5. Análisis de las soluciones obtenidas dentro de cada escenario del modelo exacto.

Escenario	Nº ejemplares solución óptima encontrada	Nº ejemplares detenidos		Nº ejemplares infactibles	% ejemplares resueltos con solución encontrada (óptima o no)
		Sin solución encontrada	Con solución encontrada		
E1: b = 0	46	1	33	0	98,75 %
E2: b = 0,1	7	32	34	7	51,25 %
E3: b = 0,2	4	30	19	27	28,75 %
E4: b = *random*	23	9	27	21	62,50 %

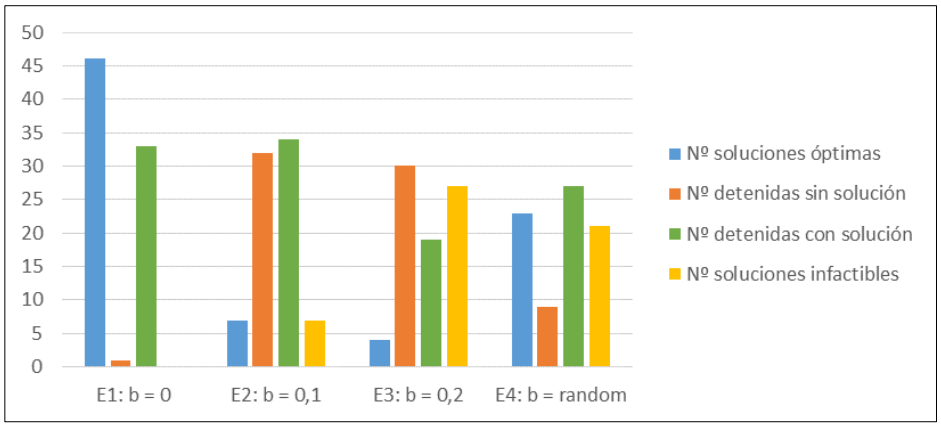


Figura 6.1. Gráfico del tipo de soluciones encontradas en cada escenario. [Fuente: propia]

Por un lado, en la Tabla 6.5 y Figura 6.1 se puede observar que cuando no se considera el deterioro lineal en las tareas, se pueden resolver prácticamente todos los ejemplares y, además, en más de la mitad de los casos obteniendo el óptimo. En cambio, cuando se empieza a considerar el deterioro, este porcentaje se reduce bastante y, en consecuencia, se hace más difícil encontrar una solución.

Por otro lado, comparando entre sí los escenarios con deterioro, se ve que el peor de los casos es el escenario 3. Luego, si se analiza el porcentaje de resueltos el mejor es el cuarto, ya que se logra obtener solución en más de la mitad de los 80 ejemplares testeados. Esto es debido a que como cada tarea tiene una tasa distinta comprendida entre 0,01 y 0,2, hay casos

en los que esta será muy cercana a 0 y por tanto, facilitará su resolución. Por el contrario, en el 26% de los casos no se tiene solución, ya que se considera que no existe.

Entonces, mirando los resultados globales se considera que el escenario 2 es mejor que el escenario 4, porque solo el 9% de las soluciones que genera son inviables y el resto, más de la mitad se resuelven. El 40% restante no ha dado solución por el límite de tiempo, pero esto no quiere decir que no exista, de hecho, es posible que si se dejará más tiempo en su resolución se podría acabar encontrando una.

También se comprueba en la Tabla 6.6 que, lógicamente, cuando se detiene una solución el tiempo medio de resolución es 3.600 segundos, ya que es el límite definido; y que en los escenarios que se detecta que hay ejemplares en los que no existe una solución, el tiempo medio es inferior a un minuto, y por tanto, el modelo matemático es lo suficientemente rápido para analizar cuando una solución es infactible y cuando no. En el caso de soluciones óptimas, se comprueba de nuevo que el escenario 1 es el mejor, ya que estas se logran muy rápidamente. En cuanto a los escenarios 2 y 3, estos son los más lentos respectivamente, necesitando aproximadamente media hora.

Tabla 6.6. Tiempo medio necesario para cada tipo de solución obtenida en cada escenario.

Escenario	Tiempo medio solución óptima (s)	Tiempo medio solución detenida (s)	Tiempo media solución inviable (s)
E1: b = 0	67	3.600	-
E2: b = 0,1	1.065	3.600	0
E3: b = 0,2	1.753	3.600	37
E4: b = *random*	662	3.600	0

Ahora bien, analizando los resultados recogidos en la Tabla 6.7, se puede ver que a medida que aumenta la dificultad de la categoría se van encontrando cada vez menos soluciones óptimas demostradas y hay más ejemplares infactibles. Esto se debe a que va aumentando la probabilidad de encontrar soluciones no óptimas.

Por tanto, se comprueba que en la categoría *Less tricky* y *Tricky* es donde más soluciones óptimas se obtienen, y por el contrario, en *Open* donde menos, ya que aquí se desconoce la solución óptima de los ejemplares empleados. Mientras que a medida que aumenta el *OS*, se logra un mayor número de soluciones óptimas, porque al haber más relaciones de precedencia entre las tareas es más fácil distribuirlas en las estaciones.

Tabla 6.7. Tipo de solución obtenida por tipo de categoría y OS.

Categoría	Nº soluciones óptimas			Nº soluciones detenidas			Nº ejemplares infactibles		
	OS=0,2	OS=0,6	OS=0,9	OS=0,2	OS=0,6	OS=0,9	OS=0,2	OS=0,6	OS=0,9
Less tricky	5	9	11	15	15	9	-	-	-
Tricky	8	5	13	16	15	7	-	-	-
Very tricky	5	3	13	19	11	7	-	6	-
Extremly tricky	-	2	4	14	13	8	10	5	8
Open	-	-	2	12	10	14	8	14	4

Por otra parte, para analizar el número medio de estaciones obtenido en las soluciones, se han comparado los mismos ejemplares resueltos de la misma categoría en cada escenario. Se ha hecho esta diferenciación porque los ejemplares utilizados en cada categoría son distintos, y, por tanto, al no ser comunes no tiene sentido compararlos entre sí y, además, la dificultad de resolución no es la misma.

En la Tabla 6.8 se aprecia que en cada categoría la tendencia es la misma, en el escenario 1 es en el que se consigue un número medio de estaciones menor, y en el escenario 3 el peor. Esto es consecuencia de la tasa de deterioro, porque en los escenarios en los que se tiene en cuenta, las tareas tienen tiempos ociosos y es más difícil asignarlas en las estaciones, por eso se necesita un número mayor. Además, cuanto mayor es la tasa, más estaciones se necesitan; y cuanta más dificultad de categoría, menos ejemplares se resuelven y menos coinciden entre escenarios, haciendo que la media sea un valor de menos calidad.

Tabla 6.8. Número medio de estaciones obtenidas en cada escenario para cada categoría.

Escenario	*Less tricky*	*Tricky*	*Very tricky*	*Extremly tricky*	*Open*
E1: b = 0	7,25	6,86	9,20	7,00	28,00
E2: b = 0,1	9,50	9,57	11,40	8,00	39,00
E3: b = 0,2	12,25	12,14	13,40	10,00	46,00
E4: b = *random*	10,00	10,57	13,60	9,50	38,00

Por último, en la Tabla 6.9 se observa cómo afectan el *OS* y la categoría en la resolución de los ejemplares:

Tabla 6.9. Número de ejemplares resueltos por escenario, según categoría y OS.

Escenarios	OS	Less tricky	Tricky	Very tricky	Extremly tricky	Open
E1: b=0	0,2	5	6	6	6	5
	0,6	6	5	5	5	5
	0,9	5	5	5	5	5
E2: b=0,1	0,2	2	3	0	0	0
	0,6	6	5	2	2	0
	0,9	5	5	5	3	3
E3: b=0,2	0,2	0	0	0	0	0
	0,6	3	2	0	1	0
	0,9	5	5	5	1	1
E4: b=*random*	0,2	4	5	6	1	0
	0,6	6	5	2	3	0
	0,9	5	5	5	1	2
TOTAL		**52**	**51**	**41**	**28**	**21**

En conclusión, cuando no se considera el deterioro (escenario 1), el *OS* y la categoría no influyen, ya que se consiguen resolver todos los ejemplares en todos los casos. Sin embargo, al considerar el efecto de deterioro en las tareas sí que hay una relación directa, cuanto mayor es la tasa de deterioro, a medida que aumenta la categoría y disminuye el OS más cuesta encontrar una solución. Esto tiene sentido ya que en "*OS=0,2*" las tareas tienen una orden de fuerza baja de relaciones de precedencia y hay más combinaciones de cómo asignarlas, y en categorías más difíciles el porcentaje de no encontrar una solución va creciendo.

6.3.2. Método aproximado (heurístico)

A continuación, se procede a analizar los resultados alcanzados por medio del procedimiento heurístico. Hay que tener en cuenta que como con este método no se garantiza encontrar soluciones óptimas, es necesario medir la calidad de las soluciones obtenidas para determinar su validez.

Antes de empezar, hay que destacar que el detalle de todos los resultados obtenidos con este procedimiento se encuentran en el Anexo A.2 y que en las celdas de las tablas donde no aparezcan datos, es decir, que aparezcan en blanco, quiere decir que no se ha logrado tener una solución para ese caso porque alguna tarea del ejemplar necesita más de una estación para realizarse. Esto último, se explicará más adelante en detalle para poder entenderlo.

En primer lugar, se muestra en la Tabla 6.10 el porcentaje de ejemplares resueltos por cada heurística con el fin de evaluarlas y compararlas. Los resultados que se muestran son los obtenidos una vez aplicada la optimización local. Para el análisis, hay que tener en cuenta

que las heurísticas H1 y H2 se aplican a todos los escenarios; mientras que las H3, H4 y H5 solo en el escenario 4. Por tanto, en función de esto, el número total de datos computados variará.

- **Escenario 1, 2 y 3:** para cada conjunto de datos, es decir, para el conjunto mediano (n=50 tareas) y para el grande (n=100 tareas), se testean un total de 160 muestras por escenario, 80 por cada una de las dos heurísticas que se le aplican (H1 y H2).

- **Escenario 4:** para cada conjunto de datos se testea un total de 400 muestras, 80 por cada una de las 5 heurísticas.

Tabla 6.10. Número y porcentaje de ejemplares totales resueltos para cada conjunto de datos en cada heurística y escenario, en el modelo heurístico.

Escenario	Ejemplares n = 50 tareas					% res.	Ejemplares n = 100 tareas					% res.
	H1	H2	H3	H4	H5		H1	H2	H3	H4	H5	
E1: b = 0	80	80	-	-	-	**100%**	80	80	-	-	-	**100%**
E2: b = 0,1	46	40	-	-	-	**54%**	33	33	-	-	-	**41%**
E3: b = 0,2	23	17	-	-	-	**25%**	13	11	-	-	-	**15%**
E4: b = *random*	37	36	36	34	38	**46%**	27	26	21	24	27	**31%**
TOTAL res.	186	173	36	34	38		153	150	21	24	27	
% resueltos	**58%**	**54%**	**45%**	**43%**	**48%**		**48%**	**47%**	**26%**	**30%**	**34%**	

Tabla 6.11. Tiempo medio de resolución para cada conjunto de datos en escenario y heurística, expresado en segundos, en el modelo heurístico.

Escenario	Ejemplares n=50 tareas					Ejemplares n=100 tareas				
	H1	H2	H3	H4	H5	H1	H2	H3	H4	H5
E1: b = 0	39,46	47,60	-	-	-	327,73	357,28	-	-	-
E2: b = 0,1	31,87	31,78	-	-	-	408,00	140,97	-	-	-
E3: b = 0,2	10,72	12,31	-	-	-	106,37	86,26	-	-	-
E4: b = *random*	24,36	27,85	21,29	23,30	18,38	120,26	142,63	139,01	134,23	117,52

Por un lado, desde el punto de vista de análisis de escenarios, se comprueba que el escenario 1 es el único en el que todos los ejemplares testeados se han resuelto, esto se debe a que no se considera el deterioro en las tareas y esto facilita su distribución en las estaciones. En cambio, a medida que aumenta la tasa de deterioro, el porcentaje de resueltos se va reduciendo.

En los casos en los que no se ha podido resolver el ejemplar, lo que ha sucedido es que una tarea se ha asignado a una estación y su tiempo de procesamiento necesario sobrepasa el TC de la estación, de manera que necesitaría realizarse en dos estaciones distintas. Ahora bien, por definición una tarea es indivisible y, por tanto, cuando esto ocurre se detiene la resolución del ejemplar. Esto se puede deber a que no existe una solución factible (aquella que satisface todas las restricciones del problema), o bien, sí que existe pero la heurística no la encuentra.

Por otro lado, se observa que en el escenario 1, tanto para el conjunto de datos de 50 tareas como el de 100 tareas, la regla H1 es la que ofrece una mejor calidad, ya que resuelve en menor tiempo una mayor cantidad de ejemplares. Luego, cuando se considera una tasa de deterioro fija, en los escenarios 2 y 3, para los ejemplares de 50 tareas la mejor sigue siendo la H1, pero en los ejemplares de 100 tareas pasa a ser la H2. En cambio, para el escenario 4, de las cinco reglas heurísticas, en ambos casos la más eficiente es la H5. En cuestión de tiempo, no se pueden comparar los escenarios entre sí, ya que como se ha mostrado en la Tabla 6.10 no resuelven la misma cantidad de ejemplares, y por tanto, no tiene sentido.

Ahora, mirando de forma general por conjunto de datos, según la Tabla 6.10 se puede ver que se resuelven más ejemplares de 50 tareas que de 100 y, por tanto, que cuantas menos tareas haya más fácil será encontrar una solución. Además, el tiempo de resolución también será mayor a medida que haya más tareas porque se necesita más tiempo para organizarlas en las estaciones, tal y como se demuestra en la Tabla 6.11.

Por otra parte, para continuar con el análisis de los resultados también se han utilizado los criterios siguientes:

a) Comparación con un dato de referencia:

Como se desconocen los valores óptimos del problema que se está estudiando, para comprobar la validez de los resultados se coge como referencia el número mínimo de estaciones encontradas para cada ejemplar (considerando que nunca hay deterioro), según el banco de datos utilizado. De modo que en la Tabla 6.12, se puede observar el número de veces que la solución final logra alcanzar esta referencia. En el Anexo A.2, en las tablas se muestra esta referencia en la columna “Est*”.

Tabla 6.12. Número de veces que para cada conjunto de datos se llega al valor de la referencia, por escenario y heurística, en el modelo heurístico.

Escenario	Categoría	Ejemplares n=50 tareas		Ejemplares n=100 tareas	
		H1	H2	H1	H2
E1: b = 0	*Less tricky*	16	16	16	16
	Tricky	6	8	10	11
	Very tricky	5	6	9	5
	Extremly tricky	3	3	9	11
	Open	4	2	4	2
	TOTAL	**34**	**35**	**48**	**45**

Aquí, se comprueba que sólo se logra alcanzar este dato de referencia en el escenario 1. Esto es correcto ya que los ejemplares del banco de datos no incluyen el efecto de deterioro, y por tanto, el dato que ofrecen no está preparado para problemas que lo consideren. Esto significa que para el resto de los escenarios no se puede afirmar que el método utilizado sea poco efectivo, ya que el dato de referencia no se puede aplicar. Por este motivo, no se muestran en la Tabla 6.12.

Entonces, entrando en detalle solo en el escenario 1, si para cada heurística se resuelven 80 ejemplares, se puede ver que en los que tienen 100 tareas se logra en más de la mitad de los casos obtener el valor de referencia y en los de 50 tareas en menos de la mitad. Además, el tipo de categoría también afecta, ya que, si en cada una se testean 16 ejemplares, se puede ver como en la dificultad más baja, "*Less tricky*", se logra en todos los ejemplares llegar al valor de referencia, y a medida que se va aumentando la dificultad cada vez coincide en menos.

Por último, no se considera que haya prácticamente diferencia entre considerar la regla de prioridad H1 o la H2. En general, en el caso de las instancias con 50 tareas parece que es un poco mejor ordenar por mayor número de sucesoras inmediatas (H2), mientras que con 100 tareas por mayor tiempo de proceso constante (H1).

b) Comparación entre solución inicial con *greedy* y solución con optimización local:

Para ver cuántas veces se ha logrado obtener la mejor solución en cada caso, se ha elaborado la Tabla 6.13 y la Tabla 6.14. Hay que tener presente que cuando se habla de la mejor solución encontrada con cualquier procedimiento, esta no tiene por qué ser la óptima.

Entonces, se puede ver a simple vista que en cada escenario y heurística se da más veces el hecho de que con la solución inicial se logra el mismo resultado que con la optimización local, que la optimización mejore la solución inicial. Esto significa que el algoritmo *greedy* diseñado

da soluciones aceptables. Ahora bien, también se aprecia que prácticamente el 30% de las veces, se logra obtener por medio de la optimización local una solución que tenga menor número de estaciones que la inicial. Por tanto, se comprueba que al aplicar esta heurística se ayuda a minimizar el objetivo deseado.

En cuanto a las reglas heurísticas, en los tres primeros escenarios, se aprecia que la H1 es la que genera un mayor número de soluciones iniciales que consiguen la mejor solución, tanto para el conjunto de datos de 50 tareas como el de 100 tareas; en cambio, en el escenario 4, la mejor es la H5. En el caso de las soluciones con la optimización local, para los dos tipos de conjunto de datos y para todos los escenarios, la regla con la que se consigue mayor número de soluciones que mejoren la inicial es la H2.

Además, se puede apreciar la tendencia mostrada en la Tabla 6.10, y es que en el escenario 1 es donde se obtiene el mayor número de resoluciones y en el escenario 3 el menor. También sigue siendo el mejor escenario con deterioro el que tiene la tasa menor.

Tabla 6.13. Número de veces por conjunto de datos que con la solución inicial, por heurística y escenario, se obtiene la misma solución que con la optimización local.

Escenario	Ejemplares n=50 tareas					Ejemplares n=100 tareas				
	H1	H2	H3	H4	H5	H1	H2	H3	H4	H5
E1: b = 0	46	41	-	-	-	56	47	-	-	-
E2: b = 0,1	36	30	-	-	-	24	23	-	-	-
E3: b = 0,2	19	14	-	-	-	10	7	-	-	-
E4: b = *random*	31	26	31	30	32	16	17	16	19	19

Tabla 6.14. Número de veces por conjunto de datos que la solución obtenida con optimización local, por heurística y escenario, encuentra una solución mejor que la inicial.

Escenario	Ejemplares n=50 tareas					Ejemplares n=100 tareas				
	H1	H2	H3	H4	H5	H1	H2	H3	H4	H5
E1: b = 0	34	39	-	-	-	24	33	-	-	-
E2: b = 0,1	10	10	-	-	-	9	10	-	-	-
E3: b = 0,2	4	3	-	-	-	3	4	-	-	-
E4: b = *random*	6	10	5	4	6	11	9	5	5	8

c) Número de veces que la solución local mejora la solución inicial:

Con el objetivo de estudiar con más detalle si la optimización local mejora el algoritmo de *greedy*, se recoge en la Tabla 6.15 el número de ejemplares que la optimización ha mejorado.

El valor obtenido en esta tabla no coincide exactamente con el de la Tabla 6.14, porque hay algunos casos en los que la optimización ha mejorado la solución de un mismo ejemplar en más de una vez, concretamente en dos. Este caso se ha dado 8 veces en el conjunto de ejemplares de 50 tareas, y de estas todas menos una en el escenario 1, ya que al no haber tiempos ociosos entre tareas es más fácil cederlas entre estaciones. Por otra parte, en los datos con 100 tareas, solo se ha dado este fenómeno en dos ocasiones.

Ahora bien, hay que decir que mayoritariamente, por ejemplar como mucho se ha logrado mejorar la solución inicial reduciendo un número de estación. Esto se considera que es normal cuando el heurístico que genera la solución inicial es razonablemente bueno.

También se observa de nuevo que el deterioro afecta claramente y que la H1 es la regla que más favorece a la optimización en los tres primeros escenarios, y que en el escenario con tasa aleatoria (el cuarto), tanto para 50 tareas como para 100, generalmente la mejor es la H2.

Por último, se puede concluir que al aumentar el número de tareas es más difícil mejorar la solución inicial. Esto se aprecia en los tres primeros escenarios, sin embargo, en el escenario 4, se puede ver que la tendencia es que si no se considera la misma tasa de deterioro, en los ejemplares de 100 tareas se logra mejorar más veces la solución con la optimización local que en los ejemplares de 50 tareas.

Tabla 6.15. Número de ejemplares, para cada conjunto de datos, que se logran mejorar con la optimización local, partiendo de la solución inicial obtenida con greedy.

Escenario	Ejemplares n=50 tareas					Ejemplares n=100 tareas				
	H1	H2	H3	H4	H5	H1	H2	H3	H4	H5
E1: b = 0	35	43	-	-	-	25	33	-	-	-
E2: b = 0,1	10	10	-	-	-	9	11	-	-	-
E3: b = 0,2	5	3	-	-	-	3	4	-	-	-
E4: b = *random*	6	10	5	4	6	11	9	5	5	8

d) Media del número de estaciones obtenidas:

Hay que tener en cuenta que no todos los escenarios dan solución a la misma cantidad de ejemplares porque dependen del tipo de dificultad (categoría) y de si se considera el efecto de deterioro. A esto se le añade que según la regla heurística definida, se resuelven unos u otros ejemplares. Por este motivo, se hace imposible hacer una comparación conjunta entre escenarios y se decide analizar en cada escenario por separado cual es la heurística que ofrece un mejor resultado.

De este modo se evita generar medias distorsionadas y confusas, y se podrá comparar en cada caso la calidad de las heurísticas entre ellas.

Tabla 6.16. Media del número de estaciones obtenidas para cada conjunto de datos según la dificultad Less tricky.

Categoría: *Less Tricky*	**Ejemplares n=50 tareas**					**Ejemplares n=100 tareas**				
	H1	**H2**	**H3**	**H4**	**H5**	**H1**	**H2**	**H3**	**H4**	**H5**
E1: b = 0	7,19	7,19	-	-	-	14,50	14,50	-	-	-
E2: b = 0,1	9,82	10,18	-	-	-	21,78	22,56	-	-	-
E3: b = 0,2	10,20	10,40	-	-	-	22,67	24,33	-	-	-
E4: b = *random*	9,72	10,09	10,00	9,72	9,91	19,40	19,40	19,20	18,80	18,60

Tabla 6.17. Media del número de estaciones obtenidas para cada conjunto de datos según la dificultad Tricky.

Categoría: *Tricky*	**Ejemplares n=50 tareas**					**Ejemplares n=100 tareas**				
	H1	**H2**	**H3**	**H4**	**H5**	**H1**	**H2**	**H3**	**H4**	**H5**
E1: b = 0	8,56	8,44	-	-	-	15,38	15,31	-	-	-
E2: b = 0,1	11,08	11,77	-	-	-	22,00	20,82	-	-	-
E3: b = 0,2	11,17	11,67	-	-	-	23,50	23,00	-	-	-
E4: b = *random*	10,00	10,90	10,00	10,30	10,10	19,17	19,67	19,33	19,00	19,00

Tabla 6.18. Media del número de estaciones obtenidas para cada conjunto de datos según la dificultad Very tricky.

Categoría: *Very tricky*	**Ejemplares n=50 tareas**					**Ejemplares n=100 tareas**				
	H1	**H2**	**H3**	**H4**	**H5**	**H1**	**H2**	**H3**	**H4**	**H5**
E1: b = 0	13,31	13,25	-	-	-	22,00	22,25	-	-	-
E2: b = 0,1	12,29	12,29	-	-	-	29,40	30,00	-	-	-
E3: b = 0,2	11,50	12,00	-	-	-	26,00	24,00	-	-	-
E4: b = *random*	11,33	11,83	11,67	11,67	11,83	19,00	18,00	18,00	19,00	19,00

Tabla 6.19. Media del número de estaciones obtenidas para cada conjunto de datos según la dificultad Extremly tricky.

Categoría: Extremly tricky	**Ejemplares n=50 tareas**					**Ejemplares n=100 tareas**				
	H1	**H2**	**H3**	**H4**	**H5**	**H1**	**H2**	**H3**	**H4**	**H5**
E1: b = 0	26,00	26,00	-	-	-	20,25	20,38	-	-	-
E2: b = 0,1	10,25	11,25	-	-	-	28,29	27,43	-	-	-
E3: b = 0,2	9,00	9,00	-	-	-	24,00	24,33	-	-	-
E4: b = *random*	10,75	11,25	10,75	11,00	11,50	21,75	22	22,25	21,5	21,25

Tabla 6.20. Media del número de estaciones para cada conjunto de datos según la dificultad Open.

Categoría: Open	**Ejemplares n=50 tareas**					**Ejemplares n=100 tareas**				
	H1	**H2**	**H3**	**H4**	**H5**	**H1**	**H2**	**H3**	**H4**	**H5**
E1: b = 0	30,13	29,88	-	-	-	57,50	58	-	-	-
E2: b = 0,1	/ (*)	/	-	-	-	/	/	-	-	-
E3: b = 0,2	/	/	-	-	-	/	/	-	-	-
E4: b = *random*	/	/	/	/	/	/	/	/	/	/

(*) "/" significa que no se ha encontrado una solución factible

Por un lado, en el conjunto de datos de 50 tareas, se observa que en el escenario 1 parece ser que por medio de la H2 se logra obtener una estación menos que con la H1, o bien las mismas. Mientras que en los otros 3 escenarios con deterioro, resulta que la H1 es la mejor. De este modo, se demuestra que para crear la lista que establezca la secuencia de las tareas para ejecutar el algoritmo *greedy*, funciona mejor no considerar el deterioro en la ordenación.

Por otro lado, en los ejemplares con 100 tareas, la H5 es la que más consigue minimizar el número de estaciones en el escenario 4. En el resto de escenarios, no acaba de verse claro si es mejor la H1 o la H2. En ambos casos, en la categoría con mayor dificultad ("*Open*"), si se considera el deterioro no se pueden comparar las reglas heurísticas entre sí, ya que no se encuentra una solución; en cambio, sin deterioro resulta ser mejor la H1.

Finalmente, con el fin de evaluar también como afectan las características de dificultad y *OS* directamente en los ejemplares para cada escenario, se elabora la tabla siguiente:

Tabla 6.21. Ejemplares resueltos en total en las 5 heurísticas y en ambos conjuntos de datos (n=50 tareas y n=100 tareas).

Escenarios	*OS*	*Less tricky*	*Tricky*	*Very tricky*	*Extremly tricky*	*Open*
E1: b=0	0,2	10	12	12	12	12
	0,6	12	10	10	10	10
	0,9	10	10	10	10	10
E2: b=0,1	0,2	6	6	1	0	0
	0,6	12	10	5	6	0
	0,9	10	10	10	6	4
E3: b=0,2	0,2	0	0	0	0	0
	0,6	4	3	0	2	0
	0,9	10	10	9	2	0
E4: b=*random*	0,2	10	8	3	0	0
	0,6	30	24	9	15	0
	0,9	25	25	25	6	1
TOTAL		**139**	**128**	**94**	**69**	**37**

Cada dificultad significa que se tendrá más probabilidad o menos de obtener una solución óptima. En la Tabla 6.21 se muestra que en el primer escenario se resuelven todos los ejemplares de cada categoría, y por tanto, no afecta el tipo de categoría ni el nivel de *OS*. Ahora bien, en el resto de escenarios se ve que a medida que aumenta la dificultad de categoría y disminuye el OS, es más difícil resolver.

En general, también se detecta que en el escenario 4, en las tres categorías más bajas se logran más soluciones que en el resto.

6.3.3. Comparación del método exacto y método heurístico

Con el objetivo de poder evaluar de forma conjunta los dos procedimientos diseñados para la resolución del problema, a continuación, se procede a comparar los resultados obtenidos por cada uno de ellos al testear el conjunto mediano de datos, es decir, los ejemplares con 50 tareas.

En primer lugar, comparando el porcentaje de ejemplares resueltos (mirando la Tabla 6.5 y Tabla 6.10), se puede ver que cuando no se considera la tasa de deterioro, el modelo heurístico logra resolver todos los ejemplares, y por tanto, es más eficiente que el matemático. En cambio, cuando hay deterioro en los escenarios, tiende a ofrecer una mejor solución el modelo matemático.

Tabla 6.22. Porcentaje de ejemplares resueltos en cada escenario por procedimiento usado.

Escenario	Modelo matemático	Modelo heurístico
E1: b = 0	98,75%	100%
E2: b = 0,1	51,25%	54%
E3: b = 0,2	28,75%	25%
E4: b = *random*	62,50%	46%

Acto seguido, como la experiencia computacional es la misma, se analiza el tiempo de resolución medio y total que ha necesitado cada procedimiento para resolver todos los ejemplares.

Tabla 6.23. Tiempo medio de resolución de los ejemplares en cada escenario, según el procedimiento utilizado para resolver el problema.

Escenario	Modelo matemático	Modelo heurístico				
		H1	H2	H3	H4	H5
E1: b = 0	1.917,50 s	39,46 s	47,60 s	-	-	-
E2: b = 0,1	3,063,44 s	23,17 s	21,37 s	-	-	-
E3: b = 0,2	2.305,14 s	4,54 s	4,20 s	-	-	-
E4: b = *random*	1.810,37 s	18,07 s	15,68 s	14,13 s	15,71 s	14,69 s
TOTAL	**2.274,11 s**	**21,31 s**	**22,21 s**	**14,13 s**	**15,71 s**	**14,69 s**

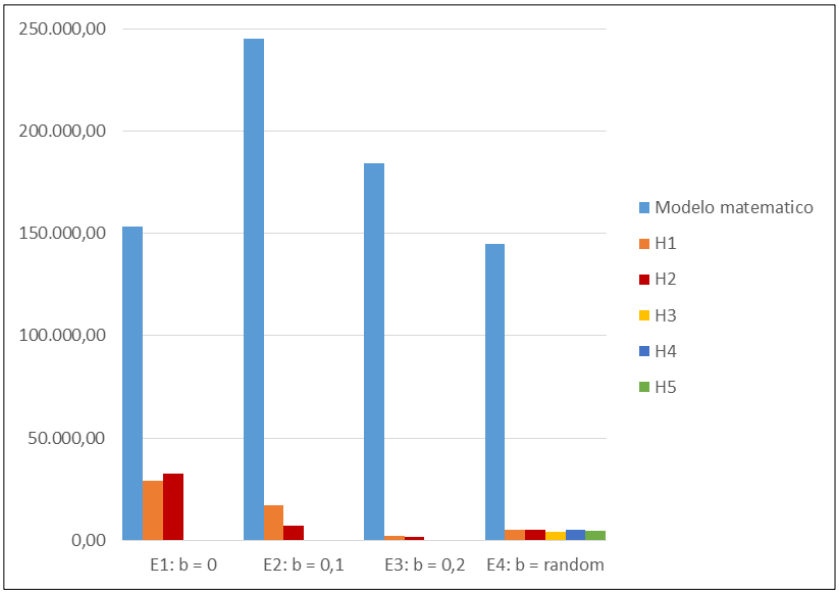


Figura 6.2. Tiempo total (en segundos) para resolver todos los ejemplares de cada escenario, según el procedimiento utilizado.

En la Tabla 6.23 y Figura 6.2. Tiempo total (en segundos) para resolver todos los ejemplares de cada escenario, según el procedimiento ut se observa que efectivamente, los métodos heurísticos son algoritmos que suelen dar, en un tiempo breve y de forma simple, una buena solución, aunque sin garantizar siempre la óptima buscada. Por este motivo, estos se suelen utilizar para acelerar el proceso de encontrar una solución satisfactoria. Sin embargo, los métodos exactos, requieren siempre de un tiempo mucho mayor pero suelen dar un mejor resultado.

En definitiva, se concluye que hay poca diferencia entre resolver los ejemplares con un procedimiento u otro, tal y como se observa en la Tabla 6.22; en cambio, en cuestión de tiempo la discrepancia es elevada. Además, parece ser que en el primer escenario la regla heurística que resuelve los ejemplares de forma más rápida es la H1, en el escenario 2 y 3 la H2, y en el escenario 4 la H3.

A continuación, con el fin de mostrar el número medio de estaciones obtenidas en cada caso, se calcula este dato a través de los mismos ejemplares resueltos en cada escenario y categoría:

Tabla 6.24. Número medio de estaciones en cada procedimiento, según categoría y escenario.

Categoría	Escenario	Modelo matemático	Modelo heurístico				
			H1	H2	H3	H4	H5
Less tricky	**E1:** b = 0	7,19	7,19	7,19	-	-	-
	E2: b = 0,1	10,83	10,42	10,50	-	-	-
	E3: b = 0,2	10,20	10,20	10,40	-	-	-
	E4: b = *random*	10,64	9,73	11,00	10,00	9,73	9,91
Tricky	**E1:** b = 0	8,00	8,57	8,44	-	-	-
	E2: b = 0,1	12,25	11,67	11,67	-	-	-
	E3: b = 0,2	9,20	10,40	10,40	-	-	-
	E4: b = *random*	11,00	10,00	10,90	10,00	10,30	10,10
Very tricky	**E1:** b = 0	12,94	13,31	13,25	-	-	-
	E2: b = 0,1	12,71	12,29	12,29	-	-	-
	E3: b = 0,2	11,50	11,50	12	-	-	-
	E4: b = *random*	13,50	11,33	11,83	11,67	11,67	11,83
Extremly tricky	**E1:** b = 0	25,06	26	26	-	-	-
	E2: b = 0,1	11,67	10,00	11,33	-	-	-
	E3: b = 0,2	8,00	9,00	9,00	-	-	-
	E4: b = *random*	10,75	10,75	11,25	10,75	11,00	11,50
Open	**E1:** b = 0	29,20	30,80	29,53	-	-	-
	E2: b = 0,1	(*)	(*)	(*)	-	-	-
	E3: b = 0,2	(*)	(*)	(*)	-	-	-
	E4: b = *random*	(*)	(*)	(*)	(*)	(*)	(*)

(*) No se pueden comparar porque no coinciden ejemplares, ya que en Open solo se resuelven prácticamente cuando no hay tasa, cuando si hay se hace inabordable el problema por tema de tiempo en la estación

En la Tabla 6.24 se puede ver que en el escenario 1, da igual que dificultad de categoría haya porque el procedimiento que obtiene menor número de estaciones siempre es el modelo matemático. Esto también ocurre en el escenario 3, donde se muestra que cuando la tasa de deterioro es alta, el procedimiento exacto consigue una media de menor. Ahora bien, tanto en el escenario 2 como en el escenario 4, al considerarse una tasa de deterioro suficientemente baja, se observa que el modelo heurístico diseñado responde mejor que el matemático, ya que logra distribuir las tareas en menos estaciones.

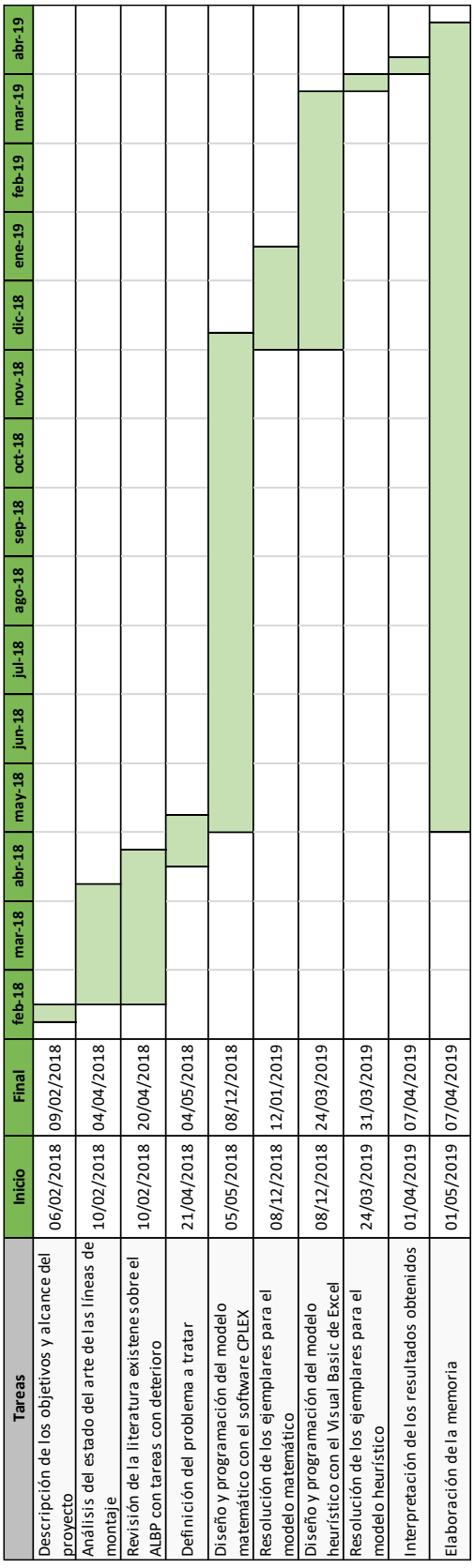
Por último, hay que destacar que en la tabla se puede ver que el número medio de estaciones se reduce a medida que hay más tasa de deterioro, pero esto no significa que se consiga una mejor solución, lo único evidente es que esto se debe a que coinciden en menor número de ejemplares. Por este motivo, se comparan los procedimientos en cada escenario por separado, ya que entre sí no se pueden extraer conclusiones de calidad.

7. Planificación y programación

La duración del presente proyecto ha sido de 1 año y 2 meses, se inició en Febrero de 2018 y se ha finalizado en Abril de 2019. A lo largo de este intervalo de tiempo, se ha llevado a cabo una serie de tareas para lograr su ejecución y consecución de forma correcta y con calidad.

Acto seguido, se presenta el diagrama de Gantt de la Figura 7.1 con el objetivo de mostrar detalladamente qué actividades se han desarrollado y cuál ha sido la planificación y programación de cada una de estas. Gracias a este cronograma, se puede observar de forma ordenada cual ha sido su duración exacta y momento de realización.

Además, hay que tener en cuenta que como el trabajo es meramente teórico y no tiene implementación práctica, después de su ejecución no hay tareas establecidas para conseguir su explotación.



Tareas	Inicio	Final
Descripción de los objetivos y alcance del proyecto	06/02/2018	09/02/2018
Análisis del estado del arte de las líneas de montaje	10/02/2018	04/04/2018
Revisión de la literatura existente sobre el ALBP con tareas con deterioro	10/02/2018	20/04/2018
Definición del problema a tratar	21/04/2018	04/05/2018
Diseño y programación del modelo matemático con el software CPLEX	05/05/2018	08/12/2018
Resolución de los ejemplares para el modelo matemático	08/12/2018	12/01/2019
Diseño y programación del modelo heurístico con el Visual Basic de Excel	08/12/2018	24/03/2019
Resolución de los ejemplares para el modelo heurístico	24/03/2019	31/03/2019
Interpretación de los resultados obtenidos	01/04/2019	07/04/2019
Elaboración de la memoria	01/05/2019	07/04/2019

Figura 7.1. Diagrama de Gantt de las tareas realizadas.

8. Impacto económico

A continuación, se procede a detallar el coste económico que ha supuesto el desarrollo del trabajo.

En primer lugar, hay que tener en cuenta que el estudiante ha compaginado este proyecto con su jornada laboral completa, y por tanto, solo se ha podido dedicar durante el fin de semana unas 4 horas de trabajo cada uno de los dos días, siendo un total de unas 8 horas semanales de media aproximadamente. Por tanto, si tal y como se ha comentado en el apartado anterior, la duración ha sido de 1 año y 2 meses y a este periodo se le descuentan 4 semanas que no se ha trabajado por vacaciones, se calcula aproximadamente un total de 52 semanas, que equivalen a 416 horas.

Por un lado, hay que añadir que para la consecución del trabajo se ha contado con la supervisión de un profesor que ha actuado como tutor y en consecuencia, esto supone más horas de dedicación a tener presente. También se consideran los costes asociados a los materiales y equipamientos utilizados.

Por otro lado, al tratarse de un estudio de investigación de carácter teórico y no práctico, ya que los procedimientos diseñados no se implementan en ninguna línea de montaje real, no hay costes asociados a la materialización del proyecto presentado.

8.1. Coste total proyecto

8.1.1. Costes de personal

En este apartado, según las horas mencionadas anteriormente, se calcula el coste asociado al trabajo realizado por un ingeniero. Este cálculo se divide en dos, por una parte, en el coste asociado a las horas que el estudiante ha dedicado al proyecto, considerándose estas como horas de junior; y por otra parte, en el coste asociado a las horas de apoyo que ha dedicado el tutor (ingeniero manager). De modo que el desglose de las horas invertidas es:

Tabla 8.1. Coste asociado a las horas dedicadas por el personal.

Concepto	Tiempo total	Coste unitario	Coste final
Ingeniero Junior (estudiante)	416 h	10 € /h	4.160 €
Ingeniero Manager (tutor)	30 h	20 € /h	600 €
		COSTE TOTAL	**4.760 €**

8.1.2. Costes de material

En lo referente a los materiales utilizados a lo largo del proyecto, se presenta seguidamente la lista detallada:

Tabla 8.2. Coste asociado a los materiales empleados.

Concepto	Tiempo total	Coste unitario	Coste final
Ordenador Lenovo G50	14 meses	48 €/ 12 meses	56 €
Software y Licencia Office	14 meses	58 €/mes	812 €
Material de oficina	-	-	20 €
		COSTE TOTAL	**888 €**

Para obtener el coste asociado al ordenador se ha calculado su amortización:

Tabla 8.3. Amortización anual según la vida útil.

Concepto por amortizar	Vida útil	Valor	Amortización[1]	Amortización anual final
Ordenador Lenovo G50	4 años	600 €	20% a 10 años	8% a 4 años

8.1.3. Costes indirectos

Por último, aunque no se pueden asignar directamente a un proceso en concreto, también hay que considerar los gastos asociados a la luz, internet y transporte que se han requerido. En el caso de la luz, a parte de las horas dedicadas por el estudiante, hay que añadir 3 semanas enteras que el ordenador estuvo encendido durante todo el día resolviendo ejemplares.

Tabla 8.4. Otros costes asociados a la ejecución del proyecto.

Concepto	Tiempo total	Coste unitario	Coste final
Luz	416 h	71,397 kW/h x 0,148 €/kWh	10,60 €
Internet	416 h	0,07 €/h	29,12 €
Transporte	40 viajes	10,20 €/ 10 viajes	40,80 €
		COSTE TOTAL	**80,52 €**

Finalmente, se muestra una tabla resumen con los tres tipos de coste que se acaba de especificar y el presupuesto total del proyecto. El importe que se presenta ya lleva el IVA incluido.

[1] Según la Tabla de coeficientes de amortización lineal de la Agencia Tributaria, para el concepto de "amortización de equipos electrónicos"

Tabla 8.5. Coste total del proyecto por tipo de concepto.

Concepto	Coste final
Costes de personal	4.760 €
Costes de material	888 €
Costes indirectos	80,52 €
COSTE FINAL	**5.729 €**

8.2. Retorno asociado al coste de reducir estación

Pese a que la resolución del problema se hace exclusivamente sobre los ejemplares obtenidos en la literatura, el objetivo es el mismo que si se aplicara en un caso real, es decir, lograr el mínimo número de estaciones necesarias en la línea de montaje.

Así pues, con el hecho de conseguir reducir aunque sea solo una estación, se podría conseguir un menor coste de recursos y de gasto energético. Esto permitiría minimizar el tiempo de producción de una unidad de producto, y en consecuencia, incrementar la productividad, permitiendo obtener mayores beneficios. Entonces, en función del coste asociado al mantenimiento de una estación y el coste total de la línea sin esa estación, se podrá concluir si se podría obtener un retorno económico. Este no tiene por qué ser el 100% del coste asociado a una estación, ya que puede ser que al quitarla se necesite de más operarios u otros gastos.

Un escenario posible sería en las cadenas de montaje donde las estaciones requeridas están especializadas en algo muy concreto, como la fabricación de un automóvil, donde el coste de mantenimiento de una sola estación es elevado. Suponiendo que fuera de 100.000€, al reducir una se podría obtener este retorno, aunque como se ha comentado, siempre teniendo en cuenta que puede ser que aumenten otros gastos, y por tanto, sea menor.

9. Impacto ambiental

La finalidad del proyecto es la creación de un modelo que permita resolver el problema de equilibrado de líneas de montaje de tipo I, considerando en las tareas el efecto del deterioro lineal. Entonces, tal y como se ha especificado en la memoria del proyecto, este trabajo es exclusivamente teórico, ya que la experiencia computacional realizada se basa en ejemplares genéricos definidos para SALBP-1.

Por tanto, al no tratarse de una simulación en una línea industrial real de la cual se puedan extraer valores que permitan evaluar el impacto ambiental, se considera que el presente proyecto no contempla emisiones de CO_2, únicamente las emisiones del ordenador portátil utilizado, teniendo en cuenta que se ha utilizado 416 horas en total:

Tabla 9.1. Emisiones de CO_2 asociadas al proyecto.

Concepto	Consumo aproximado	Factor emisión de CO_2 según consumo eléctrico[2]	Emisiones de CO_2 (kg)
Ordenador portátil	24,96 kWh	321 g CO_2/kWh	8 kg CO_2

Ahora bien, como los procedimientos diseñados sí que están pensados para utilizarlos dentro del mundo de la industria, reducir una estación permitiría necesitar menos energía requerida y esto podría significar una reducción directa de las emisiones atmosféricas de la planta de fabricación donde estuviese ubicada esta línea de ensamblaje.

[2] Valor del mix de la red eléctrica peninsular estimado en 2018. [Fuente: http://canviclimatic.gencat.cat]

Conclusiones

En este trabajo se resuelve el problema de equilibrado de líneas de montaje de tipo I, es decir, el problema de asignar las tareas con el fin de minimizar el número de estaciones necesarias, respetando un tiempo de ciclo dado. Además, se considera el efecto de deterioro lineal en las tareas para garantizar la factibilidad de la solución y obtener soluciones de buena calidad.

De manera que para llevar a cabo este objetivo, se ha diseñado un procedimiento de programación matemática, y para los ejemplares más grandes, uno heurístico. Por un lado, para facilitar la resolución del modelo matemático y evaluar la calidad de las soluciones encontradas, se han propuesto cotas. Por otro lado, en el modelo heurístico se ha decidido emplear la optimización local, partiendo de una solución inicial generada por medio de un algoritmo *greedy*, también implementado.

Para analizar los procedimientos, se han definido 4 escenarios que se diferencian por la tasa de deterioro, se han distinguido 5 tipos de dificultades en los ejemplares y se han elegido al azar un conjunto de 160 ejemplares, 80 de ellos con 50 tareas y el resto con 100 tareas. En general, de los datos obtenidos se puede extraer que considerar el efecto de deterioro en las tareas influye mucho en el resultado, ya que a medida que hay más tasa de deterioro aumenta el número de estaciones necesarias y se reduce el número de ejemplares resueltos. Esto también ocurre a medida que incrementa la dificultad (categoría) del ejemplar y cuanto más se reduce el valor de fuerza de orden del número de predecesoras entre tareas (*OS*).

También se observa que el modelo matemático ofrece un menor número de estaciones en los escenarios sin deterioro o bien con una tasa mayor de 0,1; mientras que el modelo heurístico funciona mejor en los escenarios con una tasa baja. La diferencia entre los tiempos de resolución entre ambos procedimientos es elevada, en cambio, el número medio de estaciones obtenidas es parecido. Por tanto, el método heurístico es una buena alternativa al matemático, por su rapidez de resolución y por la similitud de los resultados obtenidos.

Además, se ha podido comprobar que no hay una diferenciación significativa entre las distintas reglas heurísticas diseñadas, pero sí que la H1 logra un mayor número de soluciones iniciales buenas con el algoritmo *greedy* y que la H2 es la que obtiene un mayor número de soluciones mejoradas con la optimización local. En consecuencia, se considera que el procedimiento heurístico diseñado es suficientemente bueno.

En definitiva, se concluye que los métodos diseñados pueden ser un buen punto de partida para estudiar este problema considerando el deterioro. Ahora bien, en futuras líneas de investigación, se debería intentar mejorarlos para poder resolver los ejemplares no solucionados, así como, mejorar el resultado de los que no se ha resuelto de forma óptima. Estaría bien, intentar con otra metaheurística para salir de posibles óptimos locales.

Bibliografía

Para terminar, se presenta en este apartado todas las fuentes consultadas para el desarrollo del proyecto. Estas se dividen en dos partes, por un lado, en las referencias bibliográficas, que se corresponden a todas las citadas a lo largo de la redacción de la memoria; y por otro lado, en la bibliografía de consulta, la cual se trata de las obras no citadas explícitamente en el texto, pero que han sido necesarias para el aprendizaje del estudiante y el desarrollo del trabajo.

Referencias bibliográficas

[1] Pellini, Claudio. *Historia de la producción en serie, la cadena de montaje.* Historia y Biografías. 2015. [https://historiaybiografias.com/historia_produccion_serie/, consulta: 29 de julio de 2018].

[2] Scholl, A., & Becker, C. *State-of-the-art exact and heuristic solution procedures for simple assembly line balancing.* European Journal of Operational Research: 2006, p. 168, 666-693.

[3] Rekiek, B.; Dolgui, A.; Delchambre, A.; Bratcu, A. *State of art of optimization methods for assembly line design.* Annual Reviews in Control: 2002, p. 26, 163-174.

[4] Baybars, I. *A survey of exact algorithms for the simple assembly line balancing problem.* Management Science: 1986. Vol. 32 (8), p. 909-932.

[5] Bautista, J; Mateo, M; Ferrer, R; Pereira, J; Companys, R. *El problema de equilibrado de líneas de montaje mediante procedimientos híbridos heurísticos y de exploración dirigida.* ETSEIB (UPC). Barcelona: 2000.

[6] Scholl, A. *Balancing and sequencing of assembly lines.* Heidelberg: Physica-Verlag, 2nd ed.: 1999.

[7] Hernan, J; Medina, P.D; Cruz, E.A. *Problemas de balanceo de línea SALBP-1 y SALBP-2: Un caso de estudio.* Universidad Tecnológica de Pereira (UPC). Colombia: 2008.

[8] Jaramillo, G; Restrepo, J.H. *Aplicación de la programación dinámica para resolver el problema simple de balanceo de línea de ensamble.* Universidad Tecnológica de Pereira: 2010, p. 63-66.

[9] Talbot, F.B; Patterson, J.H., Gehrlein, W.V. *A comparative evaluation of heuristic line balancing techniques.* Management Science: 1986, Vol. 32, p. 430-454.

[10] Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA: 1989.

[11] Glover, F. *Tabu Search – Part I.* ORSA Journal on Computing: 1989, Vol. 1, nº3, p. 190-206.

[12] Morillo, D; Moreno, L; Díaz, J. *(RCPSP). Parte 2.* Universidad EAFIT. Ingeniería y Ciencia: 2014, Vol. 10, nº 20, p. 203-227.

[13] Gonge *Metodologías Analíticas y Heurísticas para la Solución del Problema de Programación de Tareas con Recursos Restringidos*, A; Ingole, D.S. *Particle Swarm Optimization Approach for Line Balancing.* International Journal of Innovative and Emerging Research in Engineering. India: 2015, Vol. 2, p. 26-29.

[14] Canizo, E; Lucero, P. *Software para programación lineal – LINGO/LINDO.* Investigación Operativa: 2002.

[15] Arcus, A. *A Computer Method of Sequencing Operations for Assembly Lines.* International Journal of Production Research: 1965, Vol. 4.

[16] Policonomics. *Curva de aprendizaje.* 2007. [https://neos-server.org/neos/, consulta: 13 de agosto de 2018].

[17] NEOS Server. *State-of-the-Art Solvers for Numerical Optimization.* University of Wisconsin, Madison: 2018 [http://policonomics.com/es/curva-aprendizaje/, consulta: 13 de agosto de 2018].

[18] Assembly Line Balancing. *Benchmark Data Sets by Otto et al. (2013).* Word Press. [https://assembly-line-balancing.de/salbp/benchmark-data-sets-2013/, consulta: 22 de diciembre de 2018].

[19] Otto, Alena; Otto, Christian; Scholl, Armin. *Systematic data generation and test design for solution algorithms on the example of SALBPG for assembly line balancing.* European Journal of Operational Research: 2013, nº 228, p. 33-45.

Bibliografía complementaria

Akpinar, S. *Large neighbourhood search algorithm for type-II assembly line balancing problem.* Pamukkale Univ Muh Bilim Derg, 23(4): 2017, p. 444-450.

Bautista-Valhondo, J. *Diseño y equilibrado de líneas de montaje. Parte I.* ETSEIB (UPC). Barcelona: 2016. Grupo de Investigación: OPE-PROTHIUS.

Bautista-Valhondo, J; Alfaro R. *Dirección de Operaciones. Proyectos singulares II.* ETSEIB (UPC). Temario Máster en Ingeniería de Organización: OPE-PROTHIUS. Barcelona: 2017.

Bahalke, U; Dolatkhahi, K; Dehghani, H; Jahani, E; Yazdanparast, V; Hajihosseini, H. *Formulation and heuristic algorithm for flow time minimization in a simple assembly line.* Journal of Engineering Manufacture 226: 2011, p. 512-526.

Baykasoglu, A; Dereli, T; Erol, R; Sabancu; I. *An ant colony based optimization algorithm for solving assembly line balancing problems.* International XII Turkish Symposium on Artificial Intelligence and Neural Networks. TAINN: 2003.

Biskup, D. *Single-machine scheduling with learning considerations.* European Journal of Operational Research: 1999, p. 173-178.

Browne, S; Yechiali, U. *Scheduling deteriorating jobs on a single processor.* Operations Research 38 (3): 1990, p. 495-498.

Burgos-Meneses, J; Garzón-Aguirre, L; López-Pereira, J. *JOMI: Algoritmo heurístico tipo greddy para solución de los problemas de líneas de ensamblaje.* Universidad de Córdoba: 2013.

Excel Total. *El Editor de Visual Basic.* En línea. [https://exceltotal.com/el-editor-de-visual-basic/, consulta: 10 de marzo de 2018 – 20 de enero de 2019].

Excel y VBA. *Curso de VBA para Excel.* En línea. [https://excelyvba.com/curso-de-vba/, consulta: 10 de marzo de 2019 – 20 de enero de 2019].

Gupta, J.N.D; Gupta, S.K. *A hybrid meta-heuristic for balancing and scheduling assembly lines with sequence-independent setup times by considering deterioration tasks and learning effect.* Scientia Iranica 21 (3): 2014, p. 963-979.

Hampta, N; Ghomi, S.M.T; Tavakkoli-Moghaddam, R; Jolai, F. *Single facility scheduling with nonlinear processing times.* Computers & Industrial Engineering 14 (4): 1988, p. 387-393.

Held, M; Karp, R.M; Shareshian, R. *Assembly line balancing-dynamic programming with*

precedence constraints. Operations Research: 1963, Vol. 11, p. 442-459.

Helgeson, W.B; Birnie, D.P. *Assembly line balancing using Ranked Positional Weight Techniques.* Journal of Industrial Engineering: 1961, Vol. 12, p. 113-159.

Hoffman, T.R. *EUREKA: A hybrid system for assembly line balancing.* Management Science: 1992, Vol. 38.

IBM. *IBM Knowledge Center*. En línea. [https://www.ibm.com/support/knowledgecenter/es/, consulta: 10 de marzo – 20 de diciembre de 2018].

Karimi-Nasab, M; Bahalke, U; Feili, H.R; Sheikhzadeh, A; Dolatkhahi, K. *Working time evaluation in assembly lines.* International Journal of Mathematics in Operational Research: 2012, Vol. 4, nº1.

Kriengkorakot, P; Kriengkorakot, N. *A Greedy Randomized Heuristic for U-Shaped Assembly Line Balancing.* Faculty of Engineering: 2008.

Moodie, C.L; Young, H.H. *A heuristic method of assembly line balancing for assumptions of constant of variable work element times.* Journal of Industrial Engineering: 1965, Vol. 16, p. 23-29.

Noushabadi, M.E; Bahalke, U; Dolatkhahi, K; Dolatkhahi, S; Makui, A. *Simple assembly line balancing problem under task deterioration.* International Journal of Industrial Engineering Computations 2: 2011, p. 538-592.

Pastor, R; Andres, C; Duran, A; Perez, M. *Tabu search algorithms for an industrial multi-product, multi-objective assembly line balancing problem, with reduction of task dispersion.* Journal of Operational Research Society: 2002, Vol. 53, p. 1317-1323.

Plans, J. *Classificació, modelització i resolució dels problemes de disseny i assignació de tasques en línies de producció.* UPC: Tesis Doctoral, 1999.

Petropoulos, D; Nearchou, A. *A particle swarm optimization algorithm for balancing assembly lines.* University of Patras. Department of Business Administration. Grecia: 2011, p. 118-129.

Lv, Qi. *Simple Assembly Line Balancing Using Particle Swarm Optimization Algorithm.* International Journal of Digital Content Technology and its Applications. China: 2011, Vol. 5, nº 6, p. 297-304.

Rubinovitz, J; Levitin, G. *Genetic algorithm for assembly line balancing.* International Journal of Production Economics: 1995, Vol. 41, p. 343-354.

Scholl, A; Klein, R. *SALOME: A bidirectional branch and bound procedure for assembly line balancing.* INFORMS Journal on Computing: 1997, Vol. 9, p. 319-334.

Shabani, M. *Balancing and scheduling U shaped assembly lines with the task deterioration effect.* Applied Mathematics in Engineering. Management and Technology: 2013, Vol. 3, p. 190-209.

Shahanaghi, K; Yolmeh, A.M; Bahalke, U. *Scheduling and balancing assembly lines with the task deterioration effect.* Journal of Engineering Manufacture: 2010, Vol. 224, p. 1145-1153.

Tongue, F.M. *Summary of heuristic line balancing procedure.* Management Science: 1960, Vol. 7, p. 21-39.

Toskari, M.D; Isleyen, S.K; Güner, E; Baykoç, O.F. *Assembly line balancing problem with deterioration tasks and learning effect.* Expert Systems with Applications. Turkey: 2010 a, Vol. 37, p. 1223-1228.

Toskari, M.D; Isleyen, S.K; Güner, E; Baykoç, O.F. *Simple assembly line balancing problem under the combinations of the effects of learning and deterioration.* Esmerald Insight. Assembly Automation. 2010 b, Vol. 30, p. 268-275.

Valero, J. *Modelos y algoritmos de PLE para el equilibrado de líneas de producción y montaje.* ETSEIB (UPC). Proyecto Final de Carrera: Barcelona, 1991.

White, W.W. *Comment on a paper by Bowman.* Operations Research: 1961, Vol. 9, p. 274-276.